Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

NORBERT ASCHEUER MATTEO FISCHETTI
MARTIN GRÖTSCHEL

Solving the Asymmetric Travelling Salesman Problem with Time Windows by Branch-and-Cut

Preprint SC 99-31 (August 1999)

Solving the Asymmetric Travelling Salesman Problem with Time Windows by Branch-and-Cut

Norbert Ascheuer [*] Matteo Fischetti [†] Martin Grötschel [‡]

Berlin, August 1999

Abstract

Many optimization problems have several equivalent mathematical models. It is often not apparent which of these models is most suitable for practical computation, in particular, when a certain application with a specific range of instance sizes is in focus. Our paper addresses the Asymmetric Travelling Salesman Problem with time windows (ATSP-TW) from such a point of view. The real–world application we aim at is the control of a stacker crane in a warehouse.

We have implemented codes based on three alternative integer programming formulations of the ATSP-TW and more than ten heuristics. Computational results for real-world instances with up to 233 nodes are reported, showing that a new model presented in a companion paper outperforms the other two models we considered — at least for our special application — and that the heuristics provide acceptable solutions.

Keywords : Asymmetric Travelling Salesman Problem, Time Windows, Integer Programs, Branch&Cut-Algorithm, Heuristics, Control of Stacker Cranes

MSC : 90C10, 90C27, 90C35

1 Introduction

This paper presents a computational study of the following time-constrained version of the asymmetric travelling salesman problem (ATSP): Consider a directed graph $D := (V \cup \{0\}, A)$ on $n+1$ nodes. Node 0 is the starting node (depot) for a salesman. With each arc $(i, j) \in A$, an arc duration $c_{ij} > 0$ is associated. Furthermore, assume that for each node $i \in V$, a processing time $p_i \geq 0$, a release date $r_i \geq 0$, and a due date $d_i \geq r_i$ are given. The release date r_i denotes the earliest possible (and the due date d_i the latest possible) starting time for visiting (processing) node $i \in V$. For the depot node 0 we assume $r_0 = d_0 = 0$. The processing time p_i represents the elapsed time between the arrival and the departure at node i. Throughout this paper we assume that arc durations, processing times, release dates, and due dates are nonnegative integer values. For due dates we also allow $d_i = \infty$. The interval $[r_i, d_i]$ is called the *time window* of node i, the *width* of the time window is given by $d_i - r_i$. The time window for node $i \in V$ is called *active*, if $r_i > 0$ or $d_i < \infty$. A time window $[0, \infty)$ is called *relaxed*.

[*]Intranetz GmbH, Bergstr. 22, 10115 Berlin, Germany, http://www.intranetz.de

[†]Dipartimento di Elettronica ed Informatica, University of Padova, Italy (work supported by C.N.R.,Italy)

[‡]Konrad–Zuse–Zentrum für Informationstechnik Berlin (ZIB), Takustr. 7, 14195 Berlin–Dahlem, Germany, http://www.zib.de

The problem is to find a sequence of the nodes (starting at the depot node 0 at time 0 and ending at node 0) with minimal cost such that for every node $i \in V$ the *arrival time* t_i at node $i \in V$ lies within the given time window $[r_i, d_i]$. In our case waiting times are allowed, i.e., one may arrive at a node $i \in V$ earlier than r_i and wait until the node is "released" at time r_i.

The ATSP-TW reduces to the ATSP if $p_i = 0, r_i = 0$, and $d_i = +\infty$ for every $i \in V$. Therefore, the ATSP-TW with general time windows is $\mathcal{NP}$–hard. Indeed, it is strongly $\mathcal{NP}$–complete to find a feasible solution for the ATSP-TW [38]. Furthermore, Tsitsiklis [43] showed that the symmetric version TSP-TW with general time windows is strongly $\mathcal{NP}$–complete, even if the underlying graph G is a path and all processing times equal 0.

It is apparent that the ATSP-TW can be formulated as a dynamic program and that it can be attacked by various branch–and–bound and other enumerative techniques of integer programming. We have choosen a polyhedral approach. In this case, the ATSP-TW is formulated as an integer linear program that is solved by a cutting plane algorithm. The cutting plane algorithm is based on cuts derived from the investigation of an associated polytope.

The reasons for this line of approach were threefold. First, the polyhedral approach to the symmetric TSP, see [1, 28, 36], and the ATSP, see [23], has been extremely successful and we hoped that this "positive behaviour" might also show up in our ATSP variant. Second, we could make use of separation algorithms that had been developed in related projects. Third, the application we focus on here, was part of a larger project where LP based techniques were the work horse of the algorithmic approach.

This paper reports part of the results of an investigation for Siemens Nixdorf Informationssysteme (SNI) aiming at the (global) optimization of a semiautomatic PC manufacturing process, see Ascheuer [3] for details. One of the individual optimization problems arising here is the task to schedule the stacker cranes of automatic warehouses. In this case the stacker crane optimization can be formulated as an online ATSP, see Ascheuer et al. [5]. We have developed several online ATSP heuristics. To check the quality of the solutions of the online heuristics it is necessary to solve (offline) ATSPs with additional constraints, in particular with time windows, see [6]. The instances from this stacker crane application, supplied by SNI, provide the basic data of the present computational study.

One problem with our line of attack was that there is no "natural" IP formulation of the ATSP-TW. There are many possibilities to model the ATSP-TW as an integer or mixed–integer linear program; and there is, at least to our knowledge, no way to tell which model holds the best computational perspectives. We have chosen to concentrate on three different models, explained in Section 3, and to let "computational experience" decide which to use in practice.

The paper is organized as follows. In Section 2 we briefly sketch related problems and work. In Section 3 we introduce the notation used throughout the paper and describe the three integer programming models we consider. In Section 4 we summarize the classes of valid inequalities that are used as cutting planes in our implementations. Section 5 is dedicated to preprocessing routines that aim at tightening the given time windows, decompose the problem instance, or fix variables. In Section 6 we briefly describe our heuristics to obtain feasible solutions. Section 7 contains a description of other implementational details of the branch&cut algorithms. In Section 8 we report extensive computational results. Some concluding remarks are given in Section 9.

2 Related work

Time constrained sequencing and routing problems arise in many practical applications. Although the ATSP-TW is a basic model in many of these applications, not much attention has been paid to it so far. In most publications, exact algorithms play a minor role; the authors concentrate on the design of heuristics, often based on local search (see [39], among others).

For the symmetric TSP, to our knowledge, a few attempts have been made to solve problems with time windows to optimality, most of them based on implicit enumeration techniques (branch-and-bound, dynamic programming). Christofides et al. [15] describe a branch-and-bound algorithm in which the lower bound computation is performed via a state–space relaxation in a dynamic programming scheme. Solutions of problem instances of up to 50 nodes with "moderately tight" time windows are reported. Baker [8] also describes a branch-and-bound algorithm where the lower bound computations are reduced to the solution of a longest path problem on an acyclic network (the dual problem of a relaxation). The algorithm performs well on problems of up to 50 nodes when only a small percentage of the time windows overlap. Dumas et al. [20] present a dynamic programming algorithm for the TSP-TW that is able to solve problems of up to 200 nodes with "fairly wide" time windows. Here reductions of the state space and the state transitions are performed that are based on the time window structure. Balas and Simonetti [12] present a new dynamic programming algorithm that can be applied to a wide class of restricted travelling salesman problems. This approach yields good results on the ATSP-TW in case that the number of overlapping time windows is small [40]. Bianco et al. [14] present a dynamic programming algorithm for the TSP with time windows and precedence constraints and present computational results for instances up to 120 nodes. For surveys on time constrained routing and scheduling problems see [18, 19], among others.

Polyhedral approaches to solve problem instances to optimality are known to work well for the precedence constrained ATSP ([7]). It is unclear whether a polyhedral approach can also handle time windows.

The ATSP-TW is related to the job–shop scheduling problem (JSSP) where one considers just one of the machines. Applegate and Cook [2] implemented a cutting plane algorithm for this problem type based on polyhedral investigations. Their computational results indicate that JSSP instances are difficult to solve – at least for this algorithmic approach.

Van Eijl [44] computationally compared two different formulations of the Delivery-Man Problem. As a by-product she obtained a branch&cut-algorithm for the TSP-TW. Computational test were performed on problem instances up to 15 nodes. She reported high running times.

It is not easy to compare the different approaches to the TSP-TW as there exist no standard benchmark problems. The instance sizes (typically expressed by the number of nodes) that can be solved depend extremely on the structure of the time windows. Moreover, in all the cases mentioned here, the authors only use randomly generated data. It is not clear what their findings mean for "real–world instances". In general, authors reporting computational experiments with the TSP-TW conclude that the case where the time windows are active for about 50 % is particularly difficult.

3 Notation and modelling

Given a node set $W \subseteq V$, let

$$A(W) := \{(i,j) \in A \mid i,j \in W\}$$

denote the set of all arcs with tail and head in W. For any two node sets $U, W \subseteq V$ let

$$(U : W) := \{(i,j) \in A \mid i \in U, j \in W\}$$

denote the set of arcs with tail in U and head in W. To simplify notation, we write $(W : j)$ and $(j : W)$ instead of $(W : \{j\})$ and $(\{j\} : W)$, respectively. Given a node set $W \subset V$, we define

$$\begin{array}{lcl} \delta^-(W) & := & \{(i,j) \in A \mid i \in V \setminus W, j \in W\}, \\ \delta^+(W) & := & \{(i,j) \in A \mid i \in W, j \in V \setminus W\}, \\ \delta(W) & := & \delta^-(W) \cup \delta^+(W). \end{array}$$

The arc set $\delta(W)$ is called a *cut*. To simplify notation we write $\delta^-(v), \delta^+(v)$, and $\delta(v)$, instead of $\delta^-(\{v\}), \delta^+(\{v\})$, and $\delta(\{v\})$, respectively.

For notational convenience, a path P consisting of the arc set $\{(v_i, v_{i+1}) \mid i = 1, \ldots, k-1\}$ is sometimes denoted by $P = (v_1, v_2, ..., v_k)$. If not stated differently, the path P is always open and *simple*, i.e., $|P| = k - 1$ and $v_i \neq v_j$ for $i \neq j$. Moreover, we let

$$[P] := \{(v_i, v_j) \in A \mid 1 \leq i < j \leq k\}$$

denote the *transitive closure* of $P = (v_1, ..., v_k)$.

The *minimal time delay* for processing node j immediately after node i is given by

$$\vartheta_{ij} := p_i + c_{ij}.$$

In the application that motivated this research the triangle inequality on ϑ is satisfied, i.e.,

$$\vartheta_{ij} \leq \vartheta_{ik} + \vartheta_{kj}, \text{ for all } i,j,k \in V. \tag{3.2}$$

If not stated differently we will assume throughout the paper that (3.2) holds.

Given a path $P = (v_1, ..., v_k)$, the *earliest arrival time* t_{v_i} at node v_i $(i = 1, ..., k)$ along P is computed as

$$\begin{array}{lcl} t_{v_1} & := & r_{v_1} \\ t_{v_i} & := & \max\{t_{v_{i-1}} + \vartheta_{v_{i-1}v_i}, r_{v_i}\} \quad \text{for } i = 2, ..., k \end{array}$$

This formula yields a *waiting time* $w_{v_i} := \max\{0, r_{v_i} - (t_{v_{i-1}} + \vartheta_{v_{i-1}v_i})\}$ which is positive whenever a node v_i is reached before its release date. If $w_i = 0$ for all $i = 2, ...k$, the path is called *minimal*. We denote by $\vartheta(P) := t_{v_k}$ the earliest arrival time at the last node of P. Notice that, for every minimal path P, one has $\vartheta(P) = r_{v_1} + \sum_{i=1}^{k-1} \vartheta_{v_i v_{i+1}}$. To simplify notation, we sometimes write $\vartheta(v_1, v_2, \ldots, v_k)$ instead of $\vartheta(P)$ for $P = (v_1, v_2, \ldots, v_k)$.

A (Hamiltonian) tour $T := (v_0, v_1, ..., v_n)$ of D starting at node $v_0 = 0$ at time $r_0 = 0$ is called *feasible* if each node is visited within its time window, i.e., $r_{v_i} \leq t_{v_i} \leq d_{v_i}$ for $i = 1, ..., n$. A path $P = (v_1, ..., v_k)$ with $2 \leq k \leq n$ is said to be *infeasible* if it does not occur as a subpath in any feasible tour. Deciding whether a given path P is feasible is clearly an $\mathcal{NP}$–complete problem, even when P contains only one node, as in this case it amounts to deciding whether a feasible tour exists. Easily checkable and obvious sufficient conditions for infeasibility are given in the following lemma.

(3.3) Lemma.
A path $P = (v_1, ..., v_k)$ is infeasible, if at least one of the following conditions holds:

(i) P violates the deadline for its last node v_k, i.e., $\vartheta(P) > d_{v_k}$.

(ii) The triangle inequality (3.2) on ϑ is satisfied and there exists a node w not covered by P such that both paths $P_1 = (w, v_1, ..., v_k)$ and $P_2 = (v_1, ..., v_k, w)$ violate the given deadline on their last node, i.e., $\vartheta(P_1) > d_{v_k}$ and $\vartheta(P_2) > d_w$.

In case condition *(ii)* above is satisfied, we say that node w cannot be *covered by (an extension of) path P*. If the triangle inequality on ϑ is not satisfied condition *(ii)* can easily be modified by considering the ϑ–shortest paths from w to v_1 and from v_k to w instead of P_1 and P_2.

Time windows induce precedences among the nodes. For example, whenever the ϑ–shortest path from j to i is longer than $d_i - r_j$ we can conclude that i has to precede j in any feasible solution. Then, let $i \prec j$ denote the fact that i has to precede j in any ATSP-TW solution and let $G_P := (V, R)$ denote the *precedence digraph* where each arc $(i,j) \in R$ represents a precedence relationship $i \prec j$. Without loss of generality we may assume G_P to be acyclic and transitively closed. Moreover, let

$$\begin{array}{rcl} \pi(v) & := & \{i \in V | (i,v) \in R\}, \\ \sigma(v) & := & \{j \in V | (v,j) \in R\}. \end{array}$$

represent the set of the *predecessors* and *successors* of a node $v \in V$, respectively. Set $\pi(X) := \cup_{v \in X} \pi(v)$ and $\sigma(X) := \cup_{v \in X} \sigma(v)$ for all $X \subseteq V$.

In the sequel we introduce three different integer programming models of the ATSP-TW, each defined on a different variable set. The first model involves binary arc variables x_{ij} as well as node variables t_i. The second model uses only binary arc variables x_{ij}, whereas the third one uses binary arc variables x_{ij} as well as integer arc variables y_{ij}. For all the models, the binary variables x_{ij} for each arc $(i,j) \in A$ can be interpreted as follows:

$$x_{ij} := \begin{cases} 1, & (i,j) \in A \text{ is used,} \\ 0, & \text{otherwise.} \end{cases}$$

A formal definition of the other variables will be given in the appropriate section.

3.1 Model 1

Miller, Tucker, and Zemlin [33] proposed to substitute the subtour elimination constraints for the TSP by a smaller class of inequalities and by introducing extra variables $t_i, i = 1, ..., n$. These inequalities offer the advantage that they can easily be modified to take further side constraints into account (see [17]). For the ATSP with time windows, the t–variables are interpreted as time variables representing the arrival times at nodes, and the corresponding *MTZ–inequalities* can be written as

$$\begin{array}{rcll} t_i + \vartheta_{ij} - (1 - x_{ij}) \cdot M & \leq t_j & & i,j = 1, ..., n, i \neq j \\ r_i \;\leq\; t_i & \leq d_i & & i = 1, ..., n. \end{array} \tag{3.4}$$

where M is a large real value. The ATSP-TW can therefore be formulated as an integer linear program as follows:

$$
\begin{array}{lllll}
 & \min c^T x & & & \\
\text{s.t.} & (1) \quad x(\delta^+(i)) & = 1 & \forall\, i \in V \cup \{0\} & \\
 & (2) \quad x(\delta^-(i)) & = 1 & \forall\, i \in V \cup \{0\} & \\
 & (3) \quad t_i + \vartheta_{ij} - (1 - x_{ij}) \cdot M & \leq t_j & \forall\, (i,j) \in A, j \neq 0 & \\
 & (4) \quad t_i & \leq d_i & \forall\, i \in V & \\
 & (5) \quad t_i & \geq r_i & \forall\, i \in V & \\
 & (6) \quad t_i & \in \mathbb{N} & \forall\, i \in V \cup \{0\} & \\
 & (7) \quad x_{ij} & \in \{0,1\} & \forall\, (i,j) \in A. &
\end{array} \tag{3.5}
$$

Here $\mathbb{N} := \{0, 1, 2, \ldots\}$ denotes the set of nonnegative integers.

We will denote by

$$P_1^{TW} := conv\{(x,t) \in \mathbb{R}^{A \times V} \mid (x,t) \text{ satisfies conditions (1)–(7) in (3.5)}\}$$

the ATSP–TW polytope based on Model 1. Note that instead of a global "big M", an individual "big M_{ij}" may be defined for each inequality in (4), satisfying

$$M_{ij} \geq d_i + p_i + c_{ij} - r_j.$$

It is easy to see that, for every feasible solution (x,t) of (1)–(7), x is the incidence vector of a feasible tour satisfying all given time windows.

This model has some disadvantages. First the MTZ–inequalities (3.4) are not very strong and they can be lifted in several ways (see Section 4). Furthermore, it is known from practical experience that a "big M"–modelling will cause computational problems. Our computations reported in Section 8 confirm this observation, even if some effort is spent on trying to reduce the big–M values involved. Even more important is that the x– and t–variables are only weakly linked via the MTZ–inequalities, i.e., the structure of the time windows has only very limited influence on the tour described by the x–variables.

3.2 Model 2

In a companion paper [4] we introduced a new model that is defined on binary arc variables only. In this model the time window restrictions are modelled by an additional class of inequalities, the so–called *infeasible path elimination constraints*. Let $x(P)$ denote the sum of the variables corresponding to a path P, i.e., $x(P) := \sum_{i=1}^{k-1} x_{v_i v_{i+1}}$.

$$
\begin{array}{lllll}
 & \min c^T x & & & \\
\text{s.t.} & (1) \quad x(\delta^+(i)) & = 1 & \forall\, i \in V \cup \{0\} & \\
 & (2) \quad x(\delta^-(i)) & = 1 & \forall\, i \in V \cup \{0\} & \\
 & (3) \quad x(A(W)) & \leq |W| - 1 & \forall\, W \subset V \cup \{0\}, 2 \leq |W| \leq n & \\
 & (4) \quad x(P) & \leq |P| - 1 = k - 2 & \forall \text{ infeasible path } P = (v_1, v_2, \ldots, v_k) & \\
 & (5) \quad x_{ij} & \in \{0,1\} & \forall\, (i,j) \in A &
\end{array} \tag{3.6}
$$

Inequalities (3.6)(4) forbid infeasible paths, i.e., paths violating the given time windows. Therefore, each solution x of (3.6)(1)–(5) is the incidence vector of a feasible Hamiltonian tour, and vice versa. The formulation of the infeasible path elimination constraints as stated

in (3.6)(4) can be very weak. In Section 4 we present several inequalities stronger than those of type (3.6)(4).
In analogy with Model 1, we denote by

$$P_2^{TW} := conv\{x \in \mathbb{R}^A \mid x \text{ satisfies conditions (1)–(5) in (3.6)}\}$$

the convex hull of all feasible solutions of Model (3.6).

3.3 Model 3

Maffioli and Sciomachen [32] and van Eijl [44] proposed a different model avoiding the need of "big M" terms. They introduced $|A|$ additional integer arc variables y_{ij} with the property that $x_{ij} = 0$ implies $y_{ij} = 0$. If $x_{ij} = 1$ then y_{ij} denotes the time when the processing of node i is started and indicates that node j is processed after node i. ATSP-TW can then be formulated as follows:

$$\begin{array}{llllll}
 & & \min c^T x & & & \\
\text{s.t.} & (1) & x(\delta^+(i)) & = 1 & \forall\, i \in V \cup \{0\} & \\
 & (2) & x(\delta^-(i)) & = 1 & \forall\, i \in V \cup \{0\} & \\
 & (3) & \sum\limits_{\substack{i=1\\ i\neq j}}^{n} y_{ij} + \sum\limits_{\substack{i=0\\ i\neq j}}^{n} \vartheta_{ij} \cdot x_{ij} & \leq \sum\limits_{\substack{k=0\\ k\neq j}}^{n} y_{jk} & \forall\, j \in V & (3.7)\\
 & (4) & r_i \cdot x_{ij} \;\leq\; y_{ij} & \leq d_i \cdot x_{ij} & i,j = 0,\ldots,n, i \neq j,\, i \neq 0 & \\
 & (5) & x_{ij} & \in \{0,1\} & \forall\, (i,j) \in A &
\end{array}$$

As for the previous models we denote by

$$P_3^{TW} := conv\{(x,y) \in \mathbb{R}^{AxA} \mid x \text{ satisfies conditions (1)–(5) in (3.7)}\}$$

the convex hull of all feasible solutions of Model (3.7).

Of course, the projection of P_1^{TW} and P_3^{TW} on the x-variables is the polytope P_2^{TW}. However, we do not have a description of any of these polytopes by linear equations and inequalities. Our goal is to find out how the LP relaxations of these polytopes that we know help to solve the ATSP-TW computationally.

4 Classes of valid inequalities

We summarize those known classes of inequalities for P_1^{TW}, P_2^{TW}, and for P_3^{TW}, that are used in at least one of our implementations.

Infeasible Path Elimination Constraints. Inequalities of this type forbid certain subpath that are infeasible, i.e., violate the given time windows. For a given infeasible path $P = (v_1, \ldots, v_k)$ the basic version of these inequalities is $x(P) \leq |P| - 1$. There exist, however, several possibilities to strengthen these inequalities, some of which were introduced in our companion paper [4]. Here are those we use in our actual implementation.

For every infeasible simple paths $P = (v_1, \ldots, v_k)$, the *tournament constraint*

$$x([P]) := \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} x_{v_i v_j} \leq k - 2 \quad (= |P| - 1) \tag{4.8}$$

is valid for $P_i^{TW}, i = 1,2,3$. Obviously, if $A(\{v_1, ..., v_k\})$ does not contain any feasible path the inequality can be strengthened to

$$x(A(\{v_1, ..., v_k\}) \leq k - 2. \tag{4.9}$$

Given a node set $W \subseteq V$, let $\Phi[W]$ denote a generic permutation of the nodes in W. For each node set $Q := \{v_1, ..., v_{k-1}\} \subset V$ and each node $v_k \in V \setminus Q$ such that all the paths of the form $(\Phi[Q], v_k)$ are infeasible, the inequality

$$x(A(Q)) + x(Q : v_k) \leq k - 2 \quad (= |Q| - 1) \tag{4.10}$$

is valid for $P_i^{TW}, i = 1,2,3$. Note that a similar inequality can be defined for the case in which all the paths of the form $(v_1, \Phi[Q])$ are infeasible, namely

$$x(v_1 : Q) + x(A(Q)) \leq k - 2 \quad (= |Q| - 1). \tag{4.11}$$

Moreover, for each node set $S := \{v_2, ..., v_{k-1}\} \subset V$ and for any two nodes $v_1, v_k \in V \setminus S$, $v_1 \neq v_k$, such that all paths of the form $(v_1, \Phi[S], v_k)$ are infeasible, the inequality

$$x(v_1 : S) + x(A(S)) + x(S : v_k) + x_{v_1 v_k} \leq k - 2 \quad (= |S|) \tag{4.12}$$

is valid for $P_i^{TW}, i = 1,2,3$. Note that inequalities (4.9),(4.10) and (4.11) are strengthenings of the subtour elimination constraint $x(A(Q)) \leq |Q| - 1$ as well as of the tournament constraints (4.8). However, it is not easy to decide whether all the paths of the form $(\Phi[Q], v_k)$, $(v_1, \Phi[Q])$, and $(v_1, \Phi[S], v_k)$ are infeasible, as required for the validity of inequalities (4.10) – (4.12). Easily checkable sufficient conditions are given by the next lemma.

(4.13) Lemma. *Assume that the triangle condition (3.2) holds.*

(a) Take any $Q \subset V$ and $v_k \in V \setminus Q$. If

$$\min_{v_i \in Q}\{r_{v_i}\} + \sum_{v_i \in Q} \min\{\vartheta_{v_i v_j} \mid v_j \in Q \cup \{v_k\}\} > d_{v_k}$$

then every path of the form $(\Phi[Q], v_k)$ is infeasible.

(b) Take any $Q \subset V$ and $v_1 \in V \setminus Q$. If

$$r_{v_1} + \sum_{v_j \in Q} \min\{\vartheta_{v_i v_j} \mid v_i \in Q \cup \{v_1\}\} > \max_{v_i \in Q}\{d_{v_i}\}$$

then every path of the form $(v_1, \Phi[Q])$ is infeasible.

(c) Take any $S \subset V$ and $v_1, v_k \in V \setminus S, v_1 \neq v_k$. If

$$r_{v_1} + \min\{\vartheta_{v_1 v_j} \mid v_j \in S\} + \sum_{v_i \in S} \min\{\vartheta_{v_i v_j} \mid v_j \in S \cup \{v_k\}\} \quad > \quad d_{v_k}$$

then every path of the form $(v_1, \Phi[S], v_k)$ is infeasible.

A more involved generalization of tournament constraints can be obtained along the following lines. Suppose we are given a family $\mathcal{P} := \{P_1, P_2, ..., P_k\}$ of node–disjoint simple paths, and let ω be any permutation of the indices of $\mathcal{P}$. The path $P = (P_{\omega(1)}, P_{\omega(2)}, ..., P_{\omega(k)})$ is called a *concatenation* of the paths in $\mathcal{P}$. Now it may happen that the paths $P_1, P_2, ..., P_k$ are feasible in themselves but that there is no way to concatenate them in a feasible way. In this case the inequality

$$\sum_{i=1}^{k} x([P_i]) \leq \sum_{i=1}^{k} |P_i| - 1 \tag{4.14}$$

is valid for $P_i^{TW}, i = 1, 2, 3$.

Lifted t-bounds. It was observed in [17] that the bounds on the t–variables $r_i \leq t_i \leq d_i$ can be strengthened by taking other arc combinations into account. Indeed, let $a_{ji} := \max\{0, r_j - r_i + \vartheta_{ji}\}$ and $b_{ij} := \max\{0, d_i - d_j + \vartheta_{ij}\}$. Then the inequalities

$$\begin{array}{llll} (i) & r_i + \sum_{\substack{j=1 \\ i \neq j}}^{n} a_{ji} x_{ji} & \leq t_i & \forall\, i \in V \\ (ii) & d_i - \sum_{\substack{j=1 \\ i \neq j}}^{n} b_{ij} x_{ij} & \geq t_i & \forall\, i \in V \end{array} \tag{4.15}$$

are valid for P_1^{TW}.

Two–job cuts. The ATSP-TW is related to the one–machine scheduling problem with time windows, i.e., the problem of sequencing n jobs on a single machine subject to a given set of time windows. Balas [9] and Dyer and Wolsey [21] considered the case where only release dates are present. The inequalities they derived can be used for the ATSP-TW too. At present, we only use the so-called *two–job cuts* (introduced by Balas [9]) in our implementation.

Suppose $r_j < r_i + p_i$ and $r_i < r_j + p_j$. The two–job cut involving order–dependent processing times can be written as

$$(\vartheta_{ij} + r_i - r_j) \cdot t_i + (\vartheta_{ji} + r_j - r_i) \geq \vartheta_{ij} \cdot \vartheta_{ji} + r_i \cdot \vartheta_{ji} + r_j \cdot \vartheta_{ij}. \tag{4.16}$$

Violated inequalities of this class can be found by enumeration of all i and j satisfying $r_j < r_i + p_i$ and $r_i < r_j + p_j$.

TSP inequalities. Obviously, all valid inequalities for the (asymmetric) travelling salesman polytope P_n^T are also valid for the ATSP-TW. The classes of inequalities that we use in our implementation are:

- D_k^-–inequalities [25, 27, 22]

$$\sum_{j=1}^{k-1} x_{i_j i_{j+1}} + x_{i_k i_1} + 2 \sum_{j=3}^{k} x_{i_1 i_j} + \sum_{j=4}^{k} \sum_{h=3}^{j-1} x_{i_j i_h} \leq k - 1 \tag{4.17}$$

- D_k^+–inequalities [25, 27, 22]

$$\sum_{j=1}^{k-1} x_{i_j i_{j+1}} + x_{i_k i_1} + 2 \sum_{j=2}^{k-1} x_{i_j i_1} + \sum_{j=3}^{k-1} \sum_{h=2}^{j-1} x_{i_j i_h} \leq k - 1 \tag{4.18}$$

- SD-inequalities [10]
  Given a *handle* $H \subset V$, disjoint *teeth* $T_1, ..., T_t$, t odd, such that $|T_i \cap H| = 1$ and $|T_i \setminus H| = 1$, and (possibly empty) disjoint node sets S and D, where $(S \cup D) \subset V \setminus (H \cup T_1 \cup ... \cup T_t)$ and $|S| + |D| = t$ is odd, the SD-inequalities have the form:

$$x((S \cup H) : (D \cup H)) + \sum_{i=1}^{t} x(A(T_i)) \leq |H| + \frac{|S| + |D| + t - 1}{2}. \tag{4.19}$$

- 2–matching constraints [27]
  Given vertex sets $H, T_1, T_2, ..., T_k \subset V, k \geq 3$ and odd satisfying

$$\begin{array}{lll} (i) & |H \cap T_i| = 1 & \text{for } i = 1, ..., k, \\ (ii) & |T_i \setminus H| = 1 & \text{for } i = 1, ..., k, \\ (iii) & T_i \cap T_j = \emptyset & \text{for } 1 \leq i < j \leq k, \end{array}$$

  the 2–matching constraint is given by

$$x(A(H)) + \sum_{i=1}^{k} x(A(T_i)) \leq |H| + \frac{k-1}{2}.$$

SOP–inequalities. The precedence–constrained ATSP, also known as Sequential Ordering Problem (SOP), is a relaxation of the ATSP-TW. All valid inequalities for the SOP are therefore valid for the ATSP-TW. We summarize the classes of inequalities actually used in our implementation.

- Predecessor inequalities (π–inequalities) [11]
  Let $S \subseteq V, \overline{S} := V \setminus S$. Then

$$x((S \setminus \pi(S)) : (\overline{S} \setminus \pi(S))) \geq 1 \tag{4.20}$$

  is valid with respect $P_i^{TW}, i = 1, 2, 3$.

- Successor inequalities (σ–inequalities) [11]
  Let $S \subseteq V, \overline{S} := V \setminus S$. Then

$$x((\overline{S} \setminus \sigma(S)) : (S \setminus \sigma(S))) \geq 1 \tag{4.21}$$

  is valid with respect $P_i^{TW}, i = 1, 2, 3$.

- Predecessor–successor inequalities (π, σ–inequalities) [11]
  Let $X, Y \subseteq V$, s.t. $i \prec j \ \forall$ pairs $i \in X, j \in Y, W := \pi(X) \cup \sigma(Y)$. Then for all $S \subset V$, s.t. $X \subset S, Y \subset \overline{S}$

$$x((S \setminus W) : (\overline{S} \setminus W)) \geq 1 \tag{4.22}$$

  is valid with respect $P_i^{TW}, i = 1, 2, 3$.

- Precedence cycle breaking inequalities (pcb–inequalities) [11]
  Let $S_1, .., S_m \subseteq V, m \geq 2$, be disjoint node sets such that $\sigma(S_i) \cap S_{i+1} \neq \emptyset$ with $S_{m+1} := S_1$. Then

$$\sum_{i=1}^{m} x(A(S_i)) \leq \sum_{i=1}^{m} |S_i| - m - 1 \tag{4.23}$$

  is valid with respect $P_i^{TW}, i = 1, 2, 3$.

- Simple pcb–inequalities
  The precedence cycle breaking inequality in its simplest form ($m = 2$ and $|S_2| = 1$) is

$$x(A(S_1)) \leq |S_1| - 2. \tag{4.24}$$

- "Special" inequalities [11]
  Let $S_1, S_2, S_3 \subset V \setminus \{1, n\}$ be disjoint node sets, with $\sigma(S_1) \cap S_2 \neq \emptyset$, $\sigma(S_2) \cap S_3 \neq \emptyset$. The following inequalities are valid with respect to $P_i^{TW}, i = 1, 2, 3$:

$$\sum_{i=1}^{2} x(A(S_i)) + x(S_2 : S_1) \leq |S_2| + |S_1| - 2, \tag{4.25}$$

$$\sum_{i=1}^{3} x(A(S_i)) + x(S_1 : S_3) \leq |S_1| + |S_2| + |S_3| - 3. \tag{4.26}$$

Strengthened (π, σ)–inequalities. In a companion paper [4] we introduced a strengthening of the (π, σ)–inequalities originally introduced by Balas et al. [11] for the precedence–constrained ATSP.

Let X and Y be two disjoint node sets such that $i \prec j$ for all $i \in X$ and $j \in Y$, and define $W := \pi(X) \cup \sigma(Y)$. Assume that the triangle inequality (3.2) on ϑ is satisfied and define

$$\widetilde{W} := W \cup \{k \in V \setminus (X \cup Y) \mid \exists\, i \in X \text{ and } j \in Y \text{ s.t. } \vartheta(i, k, j) > d_j\}$$

and

$$Q := \{(u, v) \in \delta^+(S) \mid \exists\, i \in X \text{ and } j \in Y \text{ s.t. } \vartheta(i, u, v, j) > d_j\}.$$

Then for all $S \subset V$ such that $X \subseteq S$ and $Y \subseteq \overline{S}$ the inequality

$$x((S \setminus \widetilde{W}) : (\overline{S} \setminus \widetilde{W})) \setminus Q) \geq 1$$

is valid for P_1^{TW}, P_2^{TW}, and P_3^{TW}.

The π– and σ–inequalities introduced by Balas et al. [11] can be strengthened as well (see Ascheuer [3]). Our computational experience showed, however, that for the problem instances in our test bed no such strengthenings were possible. Therefore, we omit these inequalities here.

Strengthened MTZ-Inequalities. Desrochers and Laporte [17] observed that the MTZ–subtour elimination constraints (3.4) can be lifted by taking the reverse arcs $(j, i) \in A$ and infeasible arc combinations into account. Let $a_{ji} := \max\{\vartheta_{ji}, r_i - d_j\}$, $M_{ij} \geq d_i + \vartheta_{ij} - r_j$. Then for all $i, j = 1, ..., n, i \neq j$ the inequality

$$t_i + \vartheta_{ij} - (1 - x_{ij}) \cdot M_{ij} + (M_{ij} - \vartheta_{ij} - a_{ji}) \cdot x_{ji} \leq t_j \tag{4.27}$$

is valid for P_1^{TW}.

In addition, suppose $i, j, k \in V$ are such that $r_k + \vartheta_{ki} + \vartheta_{ij} > d_j$, and choose values M and b_{ki}, such that $M \geq \max_{ij}\{c_{ij} + c_{ji}\}$ and $b_{ki} \leq M - c_{ij} - \min\{d_k + c_{ki}, d_i\}$. Then the inequality

$$t_i + \vartheta_{ij} - (1 - x_{ij}) \cdot M + (M - \vartheta_{ij} - a_{ji}) \cdot x_{ji} + b_{ki} x_{ki} \leq t_j \tag{4.28}$$

is valid for P_1^{TW}.

In case that precedences are present, the MTZ–inequalities can be further strengthened [3]. Assume $i \prec j$. As i must be scheduled before j, we know that $t_i \leq t_j$ and, even more, that the inequality

$$t_i + \vartheta_{ij} x_{ij} \leq t_j \tag{4.29}$$

is valid. Note, that inequality (4.29) can be strengthened to

$$t_i + \vartheta_{ij} x_{ij} \leq r_j$$

in case that $d_i + \vartheta_{ij} \leq r_j$ holds.

5 Data preprocessing

As for many other combinatorial optimization problems, preprocessing is an important part of an efficient implementation. Its main aim is to construct a "tighter" equivalent formulation of the problem, such that no optimal solution of the original problem is lost and each optimal solution of the tighter problem corresponds to an optimal solution of the original problem.

For the ATSP-TW, preprocessing includes three main steps: tightening the time windows, constructing precedences among the nodes, and fixing variables permanently. In addition, we detect paths $P = (v_1, v_2)$ of length one which are infeasible due to the criteria given by Lemma (3.3). If such paths are detected, the corresponding arcs (v_1, v_2) cannot be used in any feasible solution and are therefore deleted from the feasible arc set.

5.1 Tightening of the time windows

In this section we list some criteria (see, e.g., [16, 19]) that allow us to increase the release date (resp. to decrease the due date) of certain nodes.

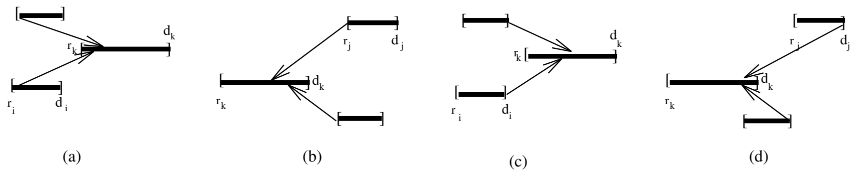

Figure 5.2

5.1.1 Release date adjustment

If the earliest arrival time at node $k \in V$ from any of its possible predecessors is bigger than its release date r_k (see Figure 5.2(a)), the release date of k can be increased, i.e.,

$$r_k := \max\{r_k, \min_{(i,k)\in A} \{r_i + \vartheta_{ik}\}\} \quad \forall k \in V \text{ s.t. } \delta^-(k) \neq \emptyset. \tag{5.30}$$

In order to avoid waiting times at the possible successor nodes of $k \in V$, the earliest possible starting time of k, and therefore its release date may be shifted (see Figure 5.2(b)), i.e.,

$$r_k := \max\{r_k, \min\{d_k, \min_{(k,j)\in A} \{r_j - \vartheta_{kj}\}\}\} \quad \forall k \in V \text{ s.t. } \delta^+(k) \neq \emptyset. \tag{5.31}$$

5.1.2 Due date adjustment

If the due date d_k of node $k \in V$ is larger than the latest possible arrival time at node k from any of its predecessors, the due date may be decreased (see Figure 5.2(c)), i.e.,

$$d_k := \min\{d_k, \max\{r_k, \max_{(i,k)\in A}\{d_i + \vartheta_{ik}\}\}\} \quad \forall k \in V \text{ s.t. } \delta^-(k) \neq \emptyset. \tag{5.32}$$

If the latest possible departure time from node $k \in V$ in order to fulfill all time window constraints for its successors is smaller than its due date, then it can be decreased (see Figure 5.2(d)), i.e.,

$$d_k := \min\{d_k, \max_{(k,j)\in A}\{d_j - \vartheta_{kj}\}\} \quad \forall k \in V \text{ s.t. } \delta^+(k) \neq \emptyset. \tag{5.33}$$

5.2 Construction of precedences

Whenever the time windows for two nodes i and j are "non–overlapping" we can infer a precedence. E.g., if $r_j + \vartheta_{ji} > d_i$ we know that node i has to precede node j in any feasible solution to the ATSP-TW. From time windows, therefore, precedences among the nodes in V can be derived and the methodology developed for the SOP [7] can also be applied to the ATSP-TW.

Let $i \prec j$ denote the fact that node i has to precede node j in every feasible solution, and let $P = (V, R)$ be the precedence digraph defined on the same node set as D and where an arc $(i, j) \in R$ represents a precedence relationship $i \prec j$. Clearly, P must be acyclic and can be assumed to be transitively closed.

5.3 Elimination of arcs

By construction, if (i, j) is in the arc set R of the precedence digraph, the arc (j, i) cannot be contained in any feasible Hamiltonian path. So we can delete all arcs (j, i) from A for which $(i, j) \in R$ holds. Furthermore, for any nodes $i, j, k \in V$ with $(i, j) \in R$ and $(j, k) \in R$, we can conclude that arc (i, k) cannot be used in any feasible Hamiltonian path as node j has to be sequenced between i and k. Therefore, those arcs (i, k) arcs are eliminated as well.

For all other arcs $(i, j) \in A$, we start with the feasible path $P = (i, j)$ and try to concatenate P with paths formed by nodes in a given node set $Q := \{v_1, ..., v_k\}$. If all these concatenations result in an infeasible path, we know that arc (i, j) cannot be used in any feasible solution and it is eliminated from the feasible arc set A. For sets Q of small cardinality this can be checked by enumerating all possible paths containing arc (i, j). In our implementation we only consider the case $| Q | \leq 2$.

Table 5.2 shows the effect these three preprocessing steps (applied iteratively several times) have on some of our instances. E.g., about 50 % of the arcs can be fixed or eliminated and lots of precedences are generated resulting in a considerable decrease of the model size.

	m	Iter.	$R1$	$R2$	TW1	TW2	TW3	TW4	Fix1	Fix2	$\|A\|$
rbg10a	110	2	27	29	0	3	0	0	0	0	54
rbg10b	110	10	32	19	25	0	5	0	30	0	41
rbg16a	272	3	94	98	0	3	4	0	0	1	79
rbg16b	272	3	54	49	0	1	1	0	0	2	167
rbg17a	306	39	62	26	235	0	4	0	74	5	174
rbg19a	380	3	154	153	0	1	3	0	0	2	71
rbg19b	380	2	90	79	0	2	1	0	0	0	211
rbg19c	380	39	74	36	245	0	4	0	72	0	229
rbg20a	420	32	158	47	141	0	10	0	198	10	95
rbg27a	756	46	142	61	438	0	3	0	92	15	487
rbg48a	2352	105	577	360	1639	0	6	0	247	0	1288
rbg49a	2450	114	734	580	1842	1	5	0	102	1	1083
rbg50a	2550	3	485	429	0	3	2	0	0	7	1629
rbg50b	2550	115	732	591	1964	1	4	0	102	0	1175
rbg50c	2550	108	611	450	1604	0	4	0	172	0	1396

Table 5.2: Preprocessing results

m	:	Number of arcs in the original input digraph
Iter.	:	Number of preprocessing loops
$R1$	:	Number of precedence relationships among the nodes
$R2$	:	Number of eliminated transitive relationships arcs
TW1	:	Number of release date adjustments due to (5.30).
TW2	:	Number of release date adjustments due to (5.31).
TW3	:	Number of deadline adjustments due to (5.32).
TW4	:	Number of deadline adjustments due to (5.33).
Fix1	:	Number of variables fixed to 0 due to the criterion described in Section 5.3 ($\|Q\| = 1$).
Fix2	:	Number of variables fixed to 0 due to the criterion described in Section 5.3 ($\|Q\| = 2$).
$\|A\|$	:	Number of remaining arcs / variables

6 Heuristics

Recall that it is an $\mathcal{NP}$-complete problem to find a feasible solution to the ATSP-TW. Therefore, unless $\mathcal{P}= \mathcal{NP}$, there is no efficient procedure that is guaranteed to terminate with such a solution. As a consequence, we run several construction heuristics varying from very simple sorting heuristics to more sophisticated insertion heuristics. Whenever a feasible solution is found we run improvement heuristics in order to obtain a better tour.

For implementational convenience we split the depot node 0 and create an additional dummy node $n+1$ such that $i \prec n+1 \;\; \forall i \in V$. Every feasible tour corresponds to a feasible Hamiltonian path starting at 0 and ending at $n+1$ in the new digraph.

6.1 Construction heuristics

Sorting heuristics:
We apply the following sorting criteria:

(1) Check if the trivial sequence $(0, 1, 2, 3, ..., n-1, n, n+1)$ is feasible.

(2) Sort the nodes according to increasing release dates and check whether this sequence is feasible.

(3) Sort the nodes according to increasing due dates and check whether this sequence is feasible.

(4) Sort the nodes according to increasing midpoints of the time windows $m_i := r_i + \frac{d_i - r_i}{2}$ and check whether this sequence is feasible.

Nearest-Feasible-Neighbor Heuristic:
Starting with each feasible arc $(0, i) \in A$ we run a nearest feasible neighbor heuristic, i.e., we enlarge the current subpath $(0, v_1, v_2, ..., v_k)$ by an arc (v_k, v_l) resulting in the smallest increase in the objective value and guaranteeing feasibility.

Insertion Heuristics:
Starting with a shortest path in A from 0 to $n + 1$ we enlarge the current partial path $P' := (0, v_1, ..., v_k, n+1)$ by a node j satisfying a certain insertion criterion. Let $W := V \setminus \{v_1, ..., v_k\}$ and $d_{min}(j) := \min\{c_{v_l j} + c_{j v_{l+1}} - c_{v_l v_{l+1}} | i \in V \setminus W, v_l \in P'$ and $(0, v_1, \ldots, v_l, j, v_{l+1}, \ldots, v_k, n+1)$ is feasible$\}$. We apply the following strategies, each resulting in a different heuristic:

(1) Among all unsequenced nodes choose the node $j \in W$ that causes the lowest increase in the path length, i.e., $d_{min}(j) = \min\{d_{min}(l) | l \in W\}$.

(2) Among all unsequenced nodes choose the node $j \in W$ that has the lowest number of feasible insertion positions and insert this node at the cheapest of these positions.

The best solution found by any of the construction heuristics is passed to the improvement heuristics.

6.2 Improvement heuristics

Swap Heuristic:
Given a feasible tour $T = (v_0, v_1, ..., v_{n+1})$ we scan through the sequence and check whether swapping two subsequent nodes v_i and $v_{i+1}, i = 1, ..., n-1$, results in a feasible solution with better objective value, in which case the swap is accepted. This procedure is repeated until no further improvement is achieved.

Two-Node-Exchange Heuristic:
This is a generalization of the Swap heuristic where any two nodes (not only subsequent nodes) in the current sequence are exchanged. If this results in a better feasible tour the exchange is accepted. This procedure is repeated until no further improvement is achieved.

Node-Reinsertion Heuristic:
Given a feasible tour $(v_0, v_1, \ldots, v_i, v_j, v_k, \ldots, v_{n+1})$ and any inner node v_j, we construct a partial tour $T' = (v_0, v_1, \ldots, v_i, v_k, \ldots, v_{n+1})$ by deleting v_j. We then try to reinsert v_j in the best position in T' such that the new sequence T is feasible. If this results in a better objective value we accept T to be the new tour and repeat until no further improvement is achieved.

Arc-Reinsertion Heuristic:
Given a feasible tour $(v_0, v_1, ..., v_i, v_j, v_k, v_l, ..., v_{n+1})$ we construct a partial tour $T' = (v_0, v_1, ..., v_i, v_l, ..., v_{n+1})$ by deleting any two consecutive nodes v_j and v_k. We then try to reinsert the pair (v_j, v_k) at any position in T' such that the new sequence T is feasible. If this results in a

better objective value we accept T to be the new tour and repeat until no further improvement is achieved.

Arc-Reversal Heuristic:
Given a feasible tour $(v_0, ..., v_j, v_k, ..., v_l, v_m, ..., v_{n+1})$ we construct a tour $T := (v_0, ..., v_j, v_l, ..., v_k, v_m, ..., v_{n+1})$ by reversing the subpath $(v_k, ..., v_l)$ such that the new sequence T is feasible. If this results in a better objective value we accept T to be the new tour and repeat until no further improvement is achieved.

Or-Exchange Heuristic: (see as well [34, 38, 41])
Given a feasible tour $(v_0, v_1, ..., v_i, ..., v_j, ..., v_{n+1})$ we remove the subpath $(v_i, ..., v_j)$ and try to reinsert it between any two subsequent nodes v_l and v_{l+1} such that the new sequence T is feasible. If this results in a better objective value we accept T to be the new tour and repeat until no further improvement is achieved. We restrict ourselves to paths involving up to 5 nodes.

The heuristics are called in the following order:

(6.34) Initial heuristics.

1. *Run Sorting Heuristics*
2. *Run Nearest Feasible Neighbor Heuristics*
3. *Run Insertion Heuristic 1*
4. *Run Insertion Heuristic 2*
5. *IF no feasible sequence found STOP*
6. *DO until no further improvement is achieved*
   (a) *Run Or-Exchange-Heuristic*
   (b) *Run Arc-Reversal Heuristic*
   (c) *Run Swap Heuristic*
   (d) *Run Arc-Reinsertion Heuristic*
   (e) *Run Node-Reinsertion Heuristic*
   (f) *Run Two-Node-exchange Heuristic*

□

The best solution found by any of the procedures is the initial feasible solution passed to the branch–and–cut code. Our computational experiments showed that these heuristics are outperformed by the LP-based heuristic procedures described in the next section.

6.3 LP–exploitation heuristic

In order to make use of the information obtained during the branch–and–cut execution, an *LP–exploitation heuristic* is run after each LP–solution. Given the current fractional point x^*, we construct the digraph $D^* := (V, A^*)$ where $(i,j) \in A^*$ if and only if $x^*_{ij} > 0$. We execute two steps.

First, in case that D^* is very sparse (in our current implementation this means that $|A^*| \leq 1.25 \cdot |V|$), we apply a branch–and–bound like implicit enumeration scheme, where we backtrack as soon as the current path becomes infeasible or the cost of the path is higher than the cost of the best feasible solution so far. We expected and observed that this procedure rarely finds a feasible tour when applied to the optimal solutions of the first LP relaxations. Feasible tours show up only at the end of the cutting plane procedure after sufficiently many inequalities have been added to the initial LP forcing out subtours, infeasible paths, and the like.

In a second step, that is always performed, we set up a modified problem instance by changing the costs c_{ij} of all $(i,j) \in A$ as follows, $c^*_{ij} := x^*_{ij} \cdot c_{ij}$. With this modified cost-matrix C^* we run all of the following construction heuristics: Nearest–Feasible–Neighbour, Insertion 1, and Insertion 2 heuristics. If a feasible solution is found we try to improve it by applying one of the following improvement heuristics: Node Reinsertion, Arc Reinsertion, Swap, and Arc Reversal. The improvement heuristics are called with the original costs c_{ij}. In order to avoid calling the improvement heuristics more than once with the same input solution we use an hash table in which the insertion key is the value of the input solution along with the index of the improvement heuristic. If the heuristic was already called with a solution of the same value, it is skipped. Note that different solutions may have the same value. Hence, we may miss a solution that leads to an improvement. Nevertheless, this strategy led to a dramatic reduction in computing time needed for the LP–exploitation heuristics.

7 The Branch&Cut Algorithm

We implemented four different branch&cut–algorithms: One for each Model (3.5)–(3.7), plus an advanced implementation based on Model 2 using the branch&cut framework ABACUS [30, 42]. We performed the advanced implementation only for this model as the preliminary version of ABACUS we had access to supported binary variables only.

We assume from now on that the preprocessing steps described in Section 5 have been performed resulting in a reduced digraph $D = (V, A)$ and tightened time windows.

7.1 Initial Linear Program

The initial linear programs for the three models are generated as follows.

Model 2. This model involves only the variables $x_{ij}, (i,j) \in A$. We generate the variables x_{ij} that correspond to arcs (i,j) in the 5–nearest–neighbours digraph and to the arcs of the best feasible tour found by the initial heuristics. (The remaining variables are later taken into account by pricing). We generate the nonnegativity constraints and the degree constraints

$$\begin{aligned} x(\delta^-(i)) &= 1 \quad \forall i \in V \cup \{0\} \\ x(\delta^+(i)) &= 1 \quad \forall i \in V \cup \{0\}. \end{aligned}$$

restricted to the initial set of x_{ij}-variables.

Model 1. We generate the same x_{ij}-variables and constraints as in Model 2. We add all node variable $t_i, i \in V$, and the corresponding lifted t–bounds (4.15). We generate, for every variable x_{ij}, the MTZ–inequality (3.5)(3), and, whenever possible, we add strengthened MTZ–inequalities of type (4.27), (4.28), (4.29).

Model 3. We generate the same x_{ij}-variables and constraints as in Model 2 and, for each x_{ij}-variable, the corresponding variable y_{ij}. We add the inequalities (3.7)(3) and (3.7)(4).

7.2 Separation routines

In this section we describe the separation procedures for the classes of inequalities listed in Section 4. For some of these classes we use routines described in the literature.

- **Subtour Elimination constraints (SEC):** We use the separation routine developed by Padberg and Rinaldi [35]. Whenever a violated SEC is found, we check if it can be strengthened to a π–, σ–, or pcb-inequality ((4.20), (4.21), or (4.23)).
- **2-matching constraints:** We use a heuristic separation procedure as described in [26].
- **π–, σ–, (π,σ)–inequalities:** We use the heuristic separation procedure for the "weak" version of these inequalities, as proposed in [11].
- **"Special" inequalities (4.25) and (4.26), PCB–inequalities (4.23):** We use the "shrinking procedure" proposed in [11]. We shrink saturated SECs both in the LP-solution digraph D^* and the precedence digraph. If an infeasible arc (i.e., a reverse or transitive arc of a precedence arc) or a cycle in the precedence digraph is detected we have a violated pcb–inequality or a violated inequality of type (4.25) or (4.26).
- **D_k–inequalities:** We use the separation procedure described in [23].
- **SD–inequalities:** We use the separation procedure described in [23].
- **Infeasible Path Elimination Constraints (IPEC):** In the test phase of our implementation we often observed the following: As soon as an IPEC is generated, the LP–solution tries to react to this cutting plane by taking a short "detour" or a "short cut". Therefore, in the final implementation we first check whether a minor modification of an already–generated infeasible path constraint is violated.

  If these trivial separation checks are not successful, tournament constraints (4.8) are separated with the help of the following simple enumeration procedure. Suppose we are given a (fractional) point x^*. It can be shown (Savelsbergh [37]) that there are only polynomially many paths P_k for which $\sum_{i=1}^{k-1}\sum_{j=i+1}^{k} x^*_{v_iv_j} - k + 2$ is greater than 0, under the assumption that $\sum_{i\in V} x^*_{ij} = \sum_{j\in V} x^*_{ij} = 1$. These paths can easily be detected by enumeration (backtrack as soon as $\sum_{i=1}^{k-1}\sum_{j=i+1}^{k} x^*_{v_iv_j} - k + 2 \le 0$).

  If a violated tournament constraint is found, we check, whether it can be lifted to an inequality of type (4.9)–(4.12).

  If no cut is found, we verify if a concatenation of paths corresponding to variables $x^*_{ij} = 1$ creates an infeasibility, i.e., if an inequality of type (4.14) is violated.
- **Strengthened (π,σ)–inequalities:**
  If $X = \{i\}$ and $Y = \{j\}$, the strengthened (π,σ)–inequality is called simple. The separation problem for the simple strengthened (π,σ)–inequalities can heuristically be solved via a separation procedure similar to the one used for the weak (π,σ)–inequalities (Balas et al. [11]). Given the LP fractional point x^*, we set up a capacitated digraph $D^* := (V, A^*)$, $A^* := \{(i,j) \in A \mid x^*_{ij} > 0\}$. To each $(i,j) \in A^*$ associate a capacity

$c^*_{ij} := x^*_{ij}$. For all $i \prec j$ we then apply the following procedure. We construct a digraph $\widetilde{D} := (\widetilde{V}, \widetilde{A})$ from D^* by deleting

- all nodes in $\pi(i) \cup \sigma(j)$,
- all nodes k such that (i,k,j) is infeasible, i.e., $r_i + \vartheta_{ik} + \vartheta_{kj} > d_j$,
- all arcs (u,v) such that (i,u,v,j) is infeasible, i.e., $r_i + \vartheta_{iu} + \vartheta_{uv} + \vartheta_{vj} > d_j$.

If we do not succeed in sending one unit of flow from i to j in $\widetilde{D}$, then the minimum capacity cut in $\widetilde{D}$ separating i from j defines a violated simple generalized (π,σ)–inequality.

- **Two-Job-Cuts:** Violated inequalities of this class can be found by enumeration of all i and j satisfying $r_j < r_i + p_i$ and $r_i < r_j + p_j$.

- **Pool-Separation:** During the run of the algorithm we maintain a *pool of active and nonactive valid inequalities.* An inequality is called active if it is both stored in the constraint matrix of the current LP and in the pool, whereas inequalities that are only stored in the pool are called nonactive. The pool is initially empty. Each generated cut is added to the constraint matrix and is stored in the pool. As soon as a constraint is nonbinding in the current LP it becomes inactive, i.e., it is removed from the constraint matrix but is still kept in the pool.

  The inequalities in the pool can be used either to reconstruct linear programs from scratch, or to check if any of the cuts generated in an earlier iteration of the algorithm is violated by the actual LP–solution (pool separation).

Separation-Order. The separation routines are called in the following order:

1. Pool Separation.
2. Subtour elimination constraints.
3. "Shrinking–procedure".
4. π–inequalities.
5. σ–inequalities.
6. Infeasible Path Elimination Constraints.
7. D_k–inequalities.
8. SD–inequalities.
9. Strengthened (π,σ)–inequalities.
10. 2–matching constraints.
11. Two–Job Cuts (only for Model 1).

Whenever one of the procedures generates a cutting plane all subsequent routines are skipped.

7.3 Variable fixing

Within the branch&cut tree (BC–tree, for short) we apply two types of variable fixings. We say that a variable x_{ij} is *fixed* to its upper or lower bound, if this operation is valid for the whole BC–tree. If this is valid only for the current BC–node and all of its children-nodes we say that this node is *set* to its upper or lower bound.

If a variable is set (resp. fixed) to zero, the corresponding arc is deleted from the current feasible arc set A. Note that fixing is a permanent operation, but due to the local character of variable settings the feasible arcs may differ between different nodes of the BC-tree.

7.3.1 Logical implications

Due to the fixing (resp. setting) of a variable, further fixings (resp. settings) may be performed. First, assume that variable x_{ij} was set (resp. fixed) to zero. If after this operation either $|\delta^+(i)| = 1$ or $|\delta^-(j)| = 1$, we know that the arc leaving i or entering j has to be used in any feasible tour. Therefore, the corresponding variable can be set (resp. fixed) to one.

Now assume, that variable x_{ij} was fixed to one. An update of the time windows is possible, i.e., we set

$$\begin{aligned} r_j &:= max\{r_j, r_i + \vartheta_{ij}\} \\ d_i &:= min\{d_i, d_j - \vartheta_{ij}\} \end{aligned}$$

and call the preprocessing routines described in Section 5. In order to guarantee that all infeasible path constraints, that are based on the time windows, remain globally valid throughout the BC-tree, this operation is only applied when a variable is fixed (not when it is set) to 1.

In addition, the following reductions can be applied:

$$\begin{array}{lll} x_{ij} = 1 \quad \Rightarrow & x_{ji} & = 0 \\ & x_{ik} & = 0 \quad \forall k \in V \setminus \{j\} \\ & x_{kj} & = 0 \quad \forall k \in V \setminus \{i\} \\ & x(j : \pi(i)) & = 0 \\ & x(\sigma(j) : i) & = 0 \\ & x(\pi(i) : \sigma(j)) & = 0 \\ & x(\pi(j) : \sigma(i)) & = 0 \end{array}$$

7.3.2 Reduced cost fixing

Nonbasic active variables can be fixed to their current value using reduced cost criteria. To this end, assume that we are given a global upper bound gub and a global lower bound glb. For a nonbasic variable x_{ij} we compute the associated reduced cost r_{ij}. In case that $x_{ij} = 0$ and $glb + r_{ij} > gub$ we are allowed to fix variable x_{ij} to zero, whereas $x_{ij} = 1$ and $glb - r_{ij} > gub$ implies that x_{ij} can be fixed to one. Using the same arguments nonbasic active variables can be set to their current value using a local lower bound and reduced costs instead of the global ones.

7.4 Further implementation details

Branching. Branching is performed on x–variables only. The branching variable x_{ij} is chosen to be one that is closest to 0.5. If there exist several such variables, one with highest cost coefficient c_{ij} is chosen.

Enumeration strategy. For the comparison of the three models, Depth–First–Search is applied as enumeration strategy. ABACUS also supports Breadth–First–Search and Best–First–Search. Computational test have shown that Best–First–Search yields slightly better results than the other strategies. Therefore, we have chosen Best–First–Search as default strategy for the advanced implementation of Model 2.

Key to Tables 8.1 and 8.2:

OPT	:	Value of optimal solution
#N	:	Number of generated nodes in the branch&cut tree
#LP	:	Number of linear programs that had to be solved
#ROWS	:	Number of rows (constraints) in linear program
#COLS	:	Number of columns (variables) in linear program
CPU(TOTAL)	:	Total CPU-time in minutes to solve problem instance to optimality
LB	:	Lower bound before branching in root node
UB	:	Upper bound before branching in root node
CPU	:	CPU-Time until first branching is necessary
LP	:	Percentage of computing time spent in LP solver
SEP	:	Percentage of computing time spent in separation routines
PRIC	:	Percentage of computing time spent in pricing operations
HEUR	:	Percentage of computing time spent in heuristic algorithms.

Pricing frequency: Nonactive variables are priced out at each 5-th LP-solution.

Tailing off: Whenever in a certain BC–node new cuts are added but the increase in the objective function is not sufficiently large, we say that "tailing off" occurs and perform a branching step. In our implementation we resort to branching whenever the last 10 LPs produced no improvement in the lower bound, or in case an improvement of only 1% has been achieved in the last 20 LPs.

8 Computational results

All implementations were done in the programming language C on a SUN SPARC Station 10. Only for the advanced implementation based on Model 2 we used a preliminary version of the general purpose branch&cut-framework ABACUS [29, 30, 31], supporting 0/1-variables only. Therefore, the comparison of the three different models was performed on ad hoc implementations we developed.

The LPs were solved using the callable library of CPLEX 4.09. The codes were tested on a set of real-life data from a joint project with industry that had the aim to minimize the unloaded travel time of a stacker crane within an automatic storage system (see Ascheuer [3] for details). This set of test problems was enlarged by instances for which we relaxed some randomly selected time windows. These datasets are based on the 27–node real-life instance `rbg027a` and are denoted by `rbg27.b.x`, where `x` gives the number of nodes for which time windows are present[1].

8.1 Comparison of the three models

In a first phase we performed computational experiments with all three models of the ATSP-TW described in Section 3. Our aim was to gather from these test runs information that would allow us to choose a "winner", i.e., a model that, for the range of problem instances we address, displays the best computational performance in practice.

[1]Information on how to download the problem instances mentioned in this chapter can be obtained via `http://www.zib.de/ascheuer/ATSPTWinstances.html`

From the test runs we have made, we have chosen to present here 11 benchmark problems that cover our range of typical instance sizes and that all three models were able to solve to optimality. The data of these test runs are, by and large, representative for our experience with these models. We first run the branch&cut algorithms as described in Section 7 from scratch. These results are summarized in Table 8.1. In a second set of runs we supplied optimum solutions as input to see how the implementations behave if the best upper bound is provided. The results are presented in Table 8.2.

We infer from Tables 8.1 and 8.2 that Model 2 is the clear winner. The implementation of Model 2 ran fastest in 16 out of 22 cases, was second in 2 and third in 4 cases. Model 3 won 6 test runs, and was third in all others, while Model 1 came in second in all but two cases where it was third.

A more detailed analysis also shows that one should not expect one model to be the best over the whole range of applications. Model 2 did do particularly well on problems of type `rbg*a` where time windows are active for all nodes (the usual case in our application). Our implementation of Model 3 works rather poorly in this problem range, but it outperforms the others when only a few time windows are active.

There are additional indicators that supported our choice of Model 2. We briefly mention a few of them. Our algorithm for Model 3 often ran into severe numerical problems. The linking constraints (3.7)(4) of this model caused problems whenever $d_i = \infty$. Indeed, in case an LP variable x_{ij} has a very small value (slightly above the zero tolerance of the LP-solver) the corresponding y_{ij}–variables are not necessarily forced to 0. Therefore, adjustments with parameter settings of the LP solver are necessary. Moreover, the primal simplex method of the LP solver we used often returned an error code indicating that the problem was solved to optimality but *indeterminate infeasibilities* have been detected (CPLEX - infeasibility type 11). Finally, the resulting LPs of Model 3 are extremely difficult to solve and about 90 % of the total computing time was spent within the LP solver.

The LP formulations of all three models differ significantly in their number of variables, number of constraints and their degree of difficulty. The LPs resulting from Model 2 seem to be the smallest and easiest for the LP solver. The numerical problems for Model 1 (big M) and Model 3 (linking constraints) result in high computing times to solve the LPs. Moreover, an optimal solution of the LP relaxation of Models 1 and 3 tends to have more fractional components, i.e., variables x^*_{ij} with $0 < x^*_{ij} < 1$.

In Tables 8.1 and 8.2 we report computational results for instances up to 69 nodes only, as only instances up to this size could be solved to optimality by the implementations of all three models. Larger instances could only be solved by Model 2.

For all these reasons we decided to work with Model 2 and develop a more advanced implementation based on this model.

							Root			Computing time (in %)			
	Opt	#N	#LP	#rows	#cols	CPU(total)	LB	UB	CPU	LP	Sep	Pric	Heur
Model 1													
rbg027a	268	25	215	350	208	0:13.37	265.25	268	0:01.24	60.2	6.7	24.1	6.1
rbg034a	403	3	56	528	380	0:05.95	401.67	403	0:04.35	70.4	4.7	16.0	3.0
rbg050a	414	7	171	746	447	0:55.67	411.51	430	0:14.64	68.5	2.5	23.4	3.5
rbg055a	814	3	81	760	508	0:22.11	814.00	815	0:16.60	52.7	4.2	14.3	23.7
rbg067a	1048	3	85	917	629	0:27.43	1046.67	1049	0:22.39	43.5	6.7	14.5	28.9
rbg27.b.04	170	139	1310	851	751	3:38.46	164.00	175	0:02.95	71.5	1.8	18.7	6.4
rbg27.b.08	206	271	2170	705	578	5:08.88	194.00	214	0:02.40	68.4	3.4	24.2	2.5
rbg27.b.10	210	169	2161	535	432	2:53.24	202.85	219	0:09.21	64.1	6.5	24.0	4.5
rbg27.b.13	224	7	153	323	214	0:10.70	221.12	231	0:07.37	69.7	5.9	17.4	4.3
rbg27.b.14	224	1	101	339	205	0:06.34	224	224	0:06.34	68.8	7.1	16.2	4.9
rbg27.b.18	238	1	54	304	163	0:02.86	238	238	0:02.86	72.0	4.9	9.8	6.6
Model 2													
rbg027a	268	25	189	113	174	0:04.90	265.50	268	0:00.65	23.9	15.5	42.7	11.2
rbg034a	403	3	38	126	248	0:01.25	401.67	403	0:01.12	28.0	10.4	30.4	9.6
rbg050a	414	3	97	217	353	0:11.21	412.84	430	0:08.04	14.1	12.5	54.5	10.0
rbg055a	814	5	72	190	418	0:05.94	812.67	815	0:03.87	15.3	9.6	39.1	19.5
rbg067a	1048	5	84	233	491	0:09.63	1046.00	1048	0:04.63	12.9	18.1	34.5	18.4
rbg27.b.04	170	645	5420	109	742	4:10.70	164.00	174	0:00.77	18.0	6.8	46.7	25.7
rbg27.b.08	206	1567	12515	141	550	8:50.66	190.67	214	0:01.06	24.8	15.1	47.4	10.0
rbg27.b.10	210	285	2783	119	511	1:36.88	202.85	233	0:03.70	27.9	15.8	43.1	10.8
rbg27.b.13	224	5	138	116	179	0:03.50	221.12	224	0:02.31	38.6	18.6	18.3	13.7
rbg27.b.14	224	1	111	129	167	0:02.90	224	224	0:02.90	34.5	13.8	29.3	13.4
rbg27.b.18	238	1	45	117	132	0:00.93	238	238	0:00.93	37.6	14.0	21.5	10.8
Model 3													
rbg027a	268	13	117	547	484	0:34.83	265.55	268	0:07.69	92.1	1.6	4.2	0.7
rbg034a	403	3	30	680	538	0:13.25	401.67	403	0:09.59	92.2	1.1	2.9	0.5
rbg050a	414	41	553	1983	1768	20:07.99	411.00	430	0:49.30	90.9	0.4	7.1	0.6
rbg055a	814	7	100	1156	970	1:43.56	809.15	815	1:03.23	88.0	0.7	2.7	7.0
rbg067a	1048	5	82	1373	1126	1:56.87	1043.15	1049	1:09.39	92.6	1.2	2.5	2.1
rbg27.b.04	170	133	988	1472	1448	2:27.55	164.00	172	0:03.78	77.1	2.0	11.0	7.3
rbg27.b.08	206	41	560	800	694	1:38.87	194.00	214	0:04.39	88.5	1.9	6.9	1.9
rbg27.b.10	210	15	361	661	554	0:57.38	197.00	233	0:03.70	87.7	2.1	6.5	2.7
rbg27.b.13	224	9	183	485	380	0:27.81	221.88	231	0:19.22	88.5	2.2	5.8	2.5
rbg27.b.14	224	1	130	466	346	0:16.23	224	224	0:16.23	87.7	2.1	6.0	2.5
rbg27.b.18	238	1	55	406	272	0:05.82	238	238	0:05.82	88.0	3.3	3.8	1.9

Table 8.1: Model comparison

							Root			Computing time (in %)			
	Opt	#N	#LP	#rows	#cols	CPU(total)	LB	UB	CPU	LP	Sep	Pric	Heur
Model 1													
rbg027a	268	25	215	350	208	0:13.39	265.25	268	0:01.26	60.0	7.7	24.4	4.8
rbg034a	403	3	56	528	380	0:05.96	401.67	403	0:04.34	70.0	4.7	16.8	3.2
rbg050a	414	43	469	1260	995	3:34.94	411.00	414	0:09.28	49.0	1.7	41.1	2.0
rbg055a	814	3	74	770	509	0:16.71	812.67	814	0:14.45	61.0	6.0	22.4	5.1
rbg067a	1048	1	60	879	581	0:14.45	1048	1048	0:14.45	42.6	13.4	19.0	15.6
rbg27.b.04	170	285	2278	756	661	6:14.25	164.00	170	0:02.25	70.8	1.8	19.6	7.1
rbg27.b.08	206	133	1071	670	579	1:52.19	193.00	206	0:01.78	64.4	4.3	26.1	3.0
rbg27.b.10	210	175	2258	586	471	3:21.93	197.00	210	0:01.84	65.5	5.9	23.4	4.3
rbg27.b.13	224	3	115	322	212	0:08.02	221.12	224	0:07.17	73.7	4.9	12.5	5.9
rbg27.b.14	224	1	101	339	205	0:06.36	224	224	0:06.36	68.1	6.6	15.7	5.7
rbg27.b.18	238	1	63	303	164	0:03.15	238	238	0:03.15	70.2	7.6	11.7	4.1
Model 2													
rbg027a	268	25	189	113	174	0:04.97	265.50	268	0:00.68	23.9	16.7	42.1	9.1
rbg034a	403	3	38	126	248	0:01.23	401.67	403	0:01.09	25.2	11.4	33.3	9.8
rbg050a	414	5	118	235	349	0:13.62	412.00	414	0:05.92	13.1	10.1	60.5	10.6
rbg055a	814	1	65	214	396	0:04.71	814	814	0:04.71	17.6	15.9	27.8	21.7
rbg067a	1048	3	52	218	504	0:05.73	1046.67	1048	0:04.83	12.0	16.2	33.7	14.7
rbg27.b.04	170	1543	11993	101	739	8:45.38	164.00	170	0:00.39	19.7	7.5	42.2	28.7
rbg27.b.08	206	775	6351	123	547	4:23.44	195.20	206	0:02.10	23.9	14.6	48.1	10.1
rbg27.b.10	210	225	2153	113	455	1:15.91	202.85	210	0:03.95	27.9	15.9	44.0	10.3
rbg27.b.13	224	7	166	107	186	0:04.28	221.12	224	0:02.87	31.1	14.0	33.9	14.5
rbg27.b.14	224	1	111	129	167	0:02.87	224	224	0:02.87	37.6	13.6	29.3	13.2
rbg27.b.18	238	1	59	111	135	0:01.16	238	238	0:01.16	37.9	18.1	18.1	10.3
Model 3													
rbg027a	268	13	117	547	484	0:34.79	265.55	268	0:07.68	92.2	1.6	4.3	0.7
rbg034a	403	3	30	680	538	0:13.24	401.67	403	0:09.59	92.7	1.1	2.9	0.3
rbg050a	414	9	271	1064	878	4:11.20	410.50	414	0:31.94	88.8	0.9	8.2	1.4
rbg055a	814	5	79	1162	934	1:17.75	809.15	814	0:58.50	90.9	0.7	3.4	4.0
rbg067a	1048	9	118	1309	1116	2:10.44	1044.17	1048	1:09.44	91.3	1.5	3.7	2.4
rbg27.b.04	170	161	1341	1373	1362	3:00.11	164.00	170	0:02.02	75.8	1.9	11.7	8.6
rbg27.b.08	206	39	637	911	818	1:50.77	193.00	206	0:04.30	87.8	2.1	7.4	2.0
rbg27.b.10	210	17	392	570	470	0:50.39	202.85	210	0:19.96	84.9	3.0	7.7	3.1
rbg27.b.13	224	7	183	471	366	0:28.24	221.12	224	0:19.13	88.2	2.2	6.2	2.3
rbg27.b.14	224	1	130	466	346	0:16.47	224	224	0:16.47	87.2	2.6	5.9	2.7
rbg27.b.18	238	1	57	408	262	0:05.74	238	238	0:05.74	88.7	3.3	3.3	1.9

Table 8.2: Model comparison (optimal solution supplied)

Key to Table 8.3:

n		: Number of nodes.
$\|A\|$		: Number of arcs after preprocessing
SOLUTION ...	OPT	: Value of an optimal solution. If the instance is not solved to optimality, the global lower bound glb and global upper bound gub are given in the form $[glb, gub]$.
	GAP	: Optimality gap in percent; calculated by $\frac{gub-glb}{glb} \cdot 100$.
ROOT ...	BOUNDS	: Lower bound rlb and upper bound rub at the root LP.
	GAP	: Optimality gap at the root of the BC-tree in percent; calculated by $\frac{rub-rlb}{rlb} \cdot 100$
	QUAL	: Quality of lower bound at the root of the BC-tree in percent; calculated by $100 - (\frac{gub-rlb}{rlb} \cdot 100)$
BC-TREE ...	#N	: Number of nodes in the branch&cut tree.
	LEV	: Depth of the branch&cut tree.
#CUTS		: Number of generated cutting planes.
#LP		: Number of linear programs that had to be solved.
CPU		: Total CPU-time in minutes to solve problem instance to optimality If the problem instance could not be solved to optimality within a certain time limit, this is marked by giving the CPU-time after which the run is stopped

Key to Table 8.4: (generated cuts)

POOL	: Number of inequalities generated from the pool / Number of calls of pool separation.
SEC	: Number of generated subtour elimination constraints (SEC)/ Number of calls of SEC separation routine.
TMC	: Number of generated 2–matching constraints. / Number of calls of 2–matching separation routine.
π	: Number of generated π inequalities/ Number of calls of π separation routine.
σ	: Number of generated σ inequalities/ Number of calls of σ separation routine.
(π, σ)	: Number of generated (π, σ) inequalities/ Number of calls of (π, σ) separation routine.
IPEC	: Number of generated infeasible path elimination constraints (IPEC) / Number of calls of IPEC separation routine.
SHRINK	: Number of inequalities generated by shrinking procedure (see Section 7.2)/ Number of calls.
D_k	: Number of generated D_k–inequalities/ Number of calls of D_k separation routine.
T_k	: Number of generated T_k–inequalities/ Number of calls of T_k separation routine.
SD	: Number of generated SD–inequalities / Number of calls of SD separation routine.

Key to Table 8.5: (generated infeasible path elimination constraints)

TOURN	: Number of generated tournament constraints (4.8).
CONCAT	: Number of generated concatenated infeasible path elimination constraints (4.14).
IP1A	: Number of generated infeasible path elimination constraints (IPEC) (4.9) by enumeration procedure.
IP1B	: Number of generated IPEC (4.9) by modification of already detected infeasible paths.
IP2A	: Number of generated IPEC (4.10) by enumeration procedure.
IP2B	: Number of generated IPEC (4.10) by modification of already detected infeasible paths.
IP3A	: Number of generated IPEC (4.11) by enumeration procedure.
IP3B	: Number of generated IPEC (4.11) by modification of already detected infeasible paths.
IP4A	: Number of generated IPEC (4.12) by enumeration procedure.
IP4B	: Number of generated IPEC (4.12) by modification of already detected infeasible paths.
TOTAL	: Total number of generated infeasible path elimination constraints.

Key to Table 8.6: (Timing statistics)

INIT	:	Computing time spent in initialization phase (in %).
LP	:	Computing time spent in LP solver (in %).
IMPROVE	:	Computing time spent in LP-exploitation and subsequent improvement heuristics (in %).
SEPARATION	:	Computing time spent in various separation procedures (in %).
PRICING	:	Computing time spent in pricing (in %).
MISC	:	Computing time spent in other parts of the program (in %).
TOTAL	:	Total CPU-time in minutes.

8.2 Advanced implementation of Model 2

The advanced implementation of the branch&cut-algorithm based on Model 2 was done using a preliminary version of the general purpose branch&cut-framework ABACUS that is explained in [29], the LP-solver CPLEX 5.0, and the FORTRAN implementations of the separation routines for SD- and D_k-inequalities by Fischetti and Toth [23].

We demonstrate here the performance of this code on a testbed of 50 problem instances varying from 12 to 233 nodes that were derived from a practical application. For all instances we allow a maximum computation time of 5 hours of CPU-time. All runs are executed with the default parameter settings described in the previous section. If an instance could not be solved to optimality within the given time limit, we list the lower and upper bounds found by the algorithm. The results are summarized in Tables 8.3–8.6. Table 8.3 gives general information on the performance of the implementation, Table 8.4 (resp. Table 8.5) on the number of generated cutting planes (resp. generated infeasible path elimination constraints). Table 8.6 summarizes the percentages of computing time spent in the various parts of the program.

There is no obvious way of measuring or predicting the difficulty of an ATSP-TW instance (for a particular code). For example, the real–world instance `rbg041a` with 43 nodes appears to be extremely difficult. Within a running time of 5 hours the code could only produce an upper and lower bound differing by 9.16%. To do that, 109402 LP runs were made, pool separation was called 105521 times resulting in the handling of more than 1 million cutting planes. The total number of different cutting planes generated was, however, only 46846. The (on the surface) not too different real–world problem `rbg067a` with 69 nodes was solved, on the other hand, to optimality within 6 seconds. Only 176 cutting planes were generated and one branching step was sufficient. In general, our code for Model 2 produced, for the instances of our application, either optimal solutions or feasible solutions within an acceptable quality guarantee.

From Table 8.3 one can derive that the heuristics embedded in our implementation often found an optimal solution in the root node of the branch&cut–tree. For these instances the optimality of the solutions was proven fast. For all instances that were solved to optimality the GAP between the the lower and upper bound before branching is relatively small. For all instances, except `rbg035a.2` and `rbg040a` it is within a 7% range, for most of the instances even smaller (see column ROOT ... GAP). The quality of the lower bound before branching is slightly better. For all instances solved to optimality (except `rbg040a`) it is at most 5% lower than the value of an optimum solution. For most of these instances even within a 1% range (see column ROOT ... QUAL).

For the instances that were not solved to optimality only minor improvements in the upper bound could be obtained in the branching phase, whereas the lower bound could be improved significantly (cmp. columns SOLUTION ... OPT and ROOT ... BOUNDS)

The largest instance (in terms of nodes) solved to optimality has 127 nodes. But one should note as well that there exist small instances (e.g., rbg019b) that were hard to solve by the branch&cut-algorithm. The smallest unsolved instance contains 43 nodes.

Table 8.4 gives, for each separation routine, the number of generated cuts in relation to the number of times this separation routine was called. For example, on instance rbg016b the separation routine for the π–inequalities was called 148 times generating 41 cutting planes in total. One sees that very seldomly violated 2–matching constraints, D_k–, and T_k–inequalities were found. Among the other separation routines the number of generated cuts is more or less evenly distributed.

Table 8.5 gives a more detailed view on the number of infeasible path elimination constraints (IPEC) that were generated. Most of the generated IPEC are tournament constraints. On some instances (e.g., rbg041a) most of the IPEC are tournament constraints or inequalities of type (4.12). Only on very few instances a large number of violated inequalities (4.9)–(4.11) were found. Surprisingly, violated inequalities of type (4.11) and (4.12) were never found by the separation heuristic checking modifications of already found infeasible paths.

			SOLUTION		ROOT			BC-TREE				
	n	$\|A\|$	OPT	GAP	BOUNDS	GAP	QUAL	#N	LEV	#CUTS	#LP	CPU
rbg010a	12	54	149	0.00	[148, 149]	0.68	99.32	2	1	22	14	0:00.12
rbg016a	18	79	179	0.00	[177, 179]	1.13	98.87	2	1	8	8	0:00.20
rbg016b	18	167	142	0.00	[133, 142]	6.77	93.66	76	13	288	329	0:08.80
rbg017.2	17	200	107	0.00	107	0.00	100.00	0	0	2	2	0:00.03
rbg017	17	122	148	0.00	[148, 150]	0.00	100.00	4	2	43	43	0:00.82
rbg017a	19	176	146	0.00	146	0.00	100.00	0	0	3	5	0:00.12
rbg019a	21	71	217	0.00	217	0.00	100.00	0	0	0	1	0:00.03
rbg019b	21	211	182	0.00	[180, 185]	2.78	98.88	820	29	623	1645	0:54.57
rbg019c	21	229	190	0.00	[182, 190]	4.39	95.56	58	9	360	325	0:08.72
rbg019d	21	156	344	0.00	[343, 344]	0.29	99.71	2	1	40	31	0:00.75
rbg020a	22	95	210	0.00	210	0.00	100.00	0	0	0	1	0:00.20
rbg021.2	21	237	182	0.00	182	0.00	100.00	0	0	32	10	0:00.22
rbg021.3	21	256	182	0.00	[178, 190]	6.74	97.75	340	17	788	869	0:27.15
rbg021.4	21	264	179	0.00	[177, 190]	7.34	98.87	72	10	189	237	0:05.82
rbg021.5	21	268	169	0.00	[167, 169]	1.20	98.80	76	10	199	264	0:06.63
rbg021.6	21	358	134	0.00	[133, 134]	0.75	99.24	2	1	55	54	0:01.38
rbg021.7	21	375	133	0.00	[128, 133]	3.91	96.09	24	6	131	166	0:04.30
rbg021.8	21	380	132	0.00	[129, 136]	5.43	97.67	254	17	369	672	0:17.40
rbg021.9	21	380	132	0.00	[128, 138]	7.81	96.87	320	15	620	948	0:26.12
rbg021	21	229	190	0.00	[182, 190]	4.40	95.60	58	9	360	325	0:08.75
rbg027a	29	487	268	0.00	[266, 268]	0.75	99.24	6	3	174	59	0:02.25
rbg031a	33	388	328	0.00	328	0.00	100.00	0	0	97	37	0:01.70
rbg033a	35	421	433	0.00	433	0.00	100.00	0	0	45	24	0:01.85
rbg034a	36	535	403	0.00	[401, 403]	0.50	99.50	2	1	48	16	0:00.98
rbg035a.2	37	940	166	0.00	[158, 215]	36.08	94.94	96	15	1253	698	1:04.80
rbg035a	37	477	254	0.00	254	0.00	100.00	0	0	121	34	0:01.83
rbg038a	40	486	466	0.00	[466, 474]	0.00	100.00	13204	40	20586	38855	70:32.23
rbg040a	42	539	386	0.00	[355, 393]	10.70	91.28	1756	26	2007	5605	12:31.82
rbg041a	43	628	[382, 417]	9.16	[361, 418]	15.79	84.49	23396	35	46846	109402	—*
rbg042a	44	762	[409, 435]	6.35	[394, 444]	12.69	89.59	22300	43	49238	99990	—*
rbg048a	50	1288	[455, 527]	15.82	[454, 527]	16.08	83.92	25222	49	103883	77604	—*
rbg049a	51	1083	[418, 501]	19.86	[408, 503]	23.28	77.20	17486	52	52679	61295	—*
rbg050a	52	1629	414	0.00	[414, 430]	0.00	100.00	6	2	392	121	0:18.62
rbg050b	52	1175	[453, 542]	19.65	[447, 548]	22.59	78.74	8600	25	30094	37337	—*
rbg050c	52	1396	[509, 536]	5.30	[507, 539]	6.31	94.28	25184	35	99795	94976	—*
rbg055a	57	765	814	0.00	[813, 814]	0.12	99.88	2	1	229	68	0:06.40
rbg067a	69	843	1048	0.00	[1047,1048]	0.10	99.90	2	1	176	56	0:05.95
rbg086a	88	927	[1049, 1052]	0.28	[1042,1062]	1.92	99.04	12208	30	7317	26088	—*
rbg092a	94	1367	[1102, 1111]	0.81	[1084,1111]	2.49	97.51	8828	39	12502	27938	—*
rbg125a	127	1824	1410	0.00	[1402,1412]	0.71	99.42	56	6	654	293	3:49.82
rbg132.2	132	3126	[1069, 1125]	5.24	[1053,1128]	7.12	93.16	4336	26	5001	13939	—*
rbg132	132	1575	[1348, 1400]	3.86	[1323,1400]	5.82	94.17	7628	32	5630	20294	—*
rbg152.3	152	6191	[1525, 1594]	4.53	[1521,1596]	4.93	95.20	2558	37	9340	8817	—*
rbg152	152	2125	[1770, 1792]	1.24	[1759,1792]	1.87	98.12	5038	28	7263	13732	—*
rbg172a	174	2837	[1787, 1897]	6.15	[1777,1897]	6.75	93.24	3434	33	7752	11139	—*
rbg193.2	193	6031	[1981, 2093]	5.65	[1969,2093]	6.29	93.70	1726	21	7542	7602	—*
rbg193	193	3050	[2388, 2452]	2.68	[2386,2452]	2.76	97.23	2790	28	6666	8254	—*
rbg201a	203	3287	[2159, 2296]	6.34	[2158,2296]	6.39	93.60	3282	35	3611	7395	—*
rbg233.2	233	7588	[2152, 2304]	7.06	[2146,2304]	7.36	92.64	1106	31	10520	5111	—*
rbg233	233	3766	[2647, 2786]	5.25	[2635,2786]	5.73	94.26	1200	25	11927	7254	—*

—*: time limit of 5 CPU hours exceeded

Table 8.3: Computational results for the advanced implementation of Model 2

Table 8.4: Number of generated cuts / Number of calls of separation routine

	Pool	SEC	Shrink	π	σ	(π, σ)	IPEC	D_k	SD	TMC	T_k
rbg010a	0/ 10	9/ 10	6/ 2	0/ 4	0/ 4	0/ 4	3/ 4	4/ 1	0/ 0	0/ 0	0/ 0
rbg016a	0/ 6	1/ 6	0/ 3	0/ 5	0/ 5	0/ 5	3/ 5	4/ 2	0/ 1	0/ 1	0/ 1
rbg016b	170/ 268	35/ 196	78/ 160	41/ 148	5/ 129	26/ 126	45/ 115	43/ 79	15/ 60	0/ 48	0/ 48
rbg017.2	0/ 1	2/ 1	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0
rbg017	5/ 36	21/ 32	8/ 17	4/ 18	2/ 16	0/ 14	6/ 14	2/ 8	0/ 6	0/ 6	0/ 6
rbg017a	0/ 3	3/ 3	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0
rbg019a	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0
rbg019b	1854/1235	42/ 748	182/ 718	132/ 663	139/ 585	7/ 496	76/ 491	8/ 429	35/ 421	2/ 396	0/ 394
rbg019c	437/ 265	34/ 161	44/ 132	46/ 127	26/ 111	19/ 104	178/ 94	7/ 46	6/ 40	0/ 36	0/ 36
rbg019d	7/ 27	16/ 21	0/ 13	6/ 13	4/ 11	0/ 9	3/ 9	11/ 7	0/ 0	0/ 0	0/ 0
rbg020a	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0
rbg021.2	1/ 8	8/ 7	8/ 1	0/ 2	0/ 2	0/ 2	16/ 2	0/ 0	0/ 0	0/ 0	0/ 0
rbg021.3	1110/ 829	47/ 514	198/ 465	98/ 425	105/ 367	28/ 318	236/ 303	35/ 245	41/ 221	0/ 196	0/ 196
rbg021.4	92/ 188	35/ 142	18/ 110	36/ 116	16/ 86	11/ 80	60/ 73	8/ 54	5/ 46	0/ 42	0/ 42
rbg021.5	117/ 201	27/ 149	26/ 127	20/ 126	15/ 106	20/ 98	30/ 87	53/ 68	8/ 43	0/ 40	0/ 40
rbg021.6	17/ 47	19/ 37	14/ 15	10/ 15	0/ 12	0/ 12	7/ 12	3/ 5	2/ 3	0/ 2	0/ 2
rbg021.7	32/ 141	34/ 116	5/ 78	24/ 88	12/ 65	0/ 57	35/ 57	10/ 36	11/ 26	0/ 20	0/ 20
rbg021.8	361/ 527	59/ 339	0/ 0	0/ 0	0/ 0	0/ 0	105/ 282	131/ 233	74/ 161	0/ 130	0/ 130
rbg021.9	379/ 755	107/ 527	0/ 0	0/ 0	0/ 0	0/ 0	204/ 422	227/ 323	82/ 213	0/ 175	0/ 175
rbg021	437/ 265	34/ 161	44/ 132	46/ 127	26/ 111	19/ 104	178/ 94	7/ 46	6/ 40	0/ 36	0/ 36
rbg027a	6/ 47	23/ 44	70/ 32	28/ 19	14/ 13	9/ 9	27/ 7	0/ 4	3/ 4	0/ 2	0/ 2
rbg031a	10/ 31	27/ 25	44/ 13	12/ 8	4/ 4	8/ 2	2/ 1	0/ 0	0/ 0	0/ 0	0/ 0
rbg033a	0/ 20	23/ 20	6/ 10	0/ 9	1/ 9	0/ 8	9/ 8	2/ 3	4/ 1	0/ 0	0/ 0
rbg034a	2/ 9	22/ 8	25/ 2	0/ 1	0/ 1	0/ 1	1/ 1	0/ 0	0/ 0	0/ 0	0/ 0
rbg035a.2	1944/ 569	74/ 311	462/ 264	372/ 216	162/ 144	157/ 76	22/ 42	0/ 27	4/ 27	0/ 24	0/ 24
rbg035a	24/ 30	28/ 23	74/ 14	11/ 7	5/ 4	2/ 3	1/ 2	0/ 0	0/ 0	0/ 0	0/ 0
rbg038a	81248/34489	1096/16854	1637/15191	361/15698	445/15434	269/15124	9854/14937	3693/9139	3176/6404	24/4739	31/4729
rbg040a	22491/4856	76/2012	368/1972	227/1897	193/1785	576/1676	252/1433	128/1199	185/1083	1/ 936	1/ 935
rbg041a	1006422/105521	1289/36360	6235/35656	3149/33779	6789/31796	3814/28497	20685/26266	1455/14910	3397/13652	26/11250	7/11233
rbg042a	581053/95321	1161/37375	9578/36463	4055/33553	5318/30993	4261/28187	16691/25922	2760/16302	5393/13941	2/10464	19/10462
rbg048a	671706/72116	3490/31948	2412/30064	1209/29901	17097/29260	24350/27718	49403/23359	2931/12031	2961/9404	1/7382	29/7381
rbg049a	458526/59903	2111/27184	1553/25875	579/25703	4108/25324	890/24525	33703/24315	3928/14426	5635/11117	117/7743	55/7686
rbg050a	85/ 105	54/ 76	226/ 56	47/ 36	35/ 16	28/ 7	2/ 2	0/ 0	0/ 0	0/ 0	0/ 0
rbg050b	496520/36709	832/14093	1194/13669	1294/13387	598/13120	2064/12913	18841/12516	1253/7143	3984/5984	6/3924	28/3920
rbg050c	383223/90871	8165/43329	8100/37556	2369/36803	17946/35185	5524/30715	44371/27956	8690/16638	4591/11388	28/8726	11/8717
rbg055a	40/ 56	47/ 48	140/ 31	1/ 14	26/ 13	14/ 3	1/ 1	0/ 0	0/ 0	0/ 0	0/ 0
rbg067a	37/ 45	42/ 37	125/ 22	0/ 2	8/ 2	0/ 1	1/ 1	0/ 0	0/ 0	0/ 0	0/ 0
rbg086a	99920/25129	254/10711	879/10603	254/10369	379/10220	439/10060	3098/9813	375/7777	1627/7419	6/6170	6/6168
rbg092a	207522/25549	412/8682	3202/8505	739/7808	746/7419	1370/7030	4775/6311	357/3687	885/3362	13/2730	3/2723
rbg125a	166/ 245	164/ 180	339/ 126	39/ 85	18/ 68	10/ 62	13/ 60	44/ 50	27/ 28	0/ 21	0/ 21
rbg132.2	133908/13344	293/3929	1570/3831	489/3593	578/3375	898/3132	550/2751	234/2281	383/2060	4/1763	2/1760
rbg132	136699/20045	309/6899	1059/6788	352/6619	245/6429	609/6271	2135/5945	234/4431	677/4229	2/3745	8/3744
rbg152.3	146064/7464	549/1357	5263/1229	1687/ 803	763/ 335	944/ 150	131/ 31	2/ 2	1/ 1	0/ 0	0/ 0
rbg152	70495/12925	385/5150	3138/4988	507/4474	314/4182	636/3984	1327/3634	269/2685	673/2436	10/1964	4/1959
rbg172a	58882/10680	433/3986	1850/3846	286/3635	278/3512	344/3387	2210/3242	577/2498	1746/2037	20/1352	8/1347
rbg193.2	135759/6659	498/1337	3171/1228	1518/ 901	1142/ 433	1088/ 156	121/ 21	4/ 3	0/ 0	0/ 0	0/ 0
rbg193	36185/7521	472/2918	2604/2746	434/2326	421/2118	655/1932	1670/1697	167/1056	237/ 902	6/ 763	0/ 761
rbg201a	20931/7121	361/2886	1485/2789	189/2634	117/2576	144/2518	376/2447	448/2109	475/1723	10/1443	6/1439
rbg233.2	136390/4465	562/ 938	7318/ 813	1043/ 499	977/ 244	560/ 75	53/ 13	7/ 5	0/ 0	0/ 0	0/ 0
rbg233	114981/6588	536/2015	3740/1850	569/1473	822/1235	1607/ 954	4370/ 500	135/ 138	148/ 61	0/ 13	0/ 13

	Tourn	Concat	IP1a	IP1b	IP2a	IP2b	IP3a	IP3b	IP4a	IP4b	Total
rbg010a	1	0	0	0	0	0	0	0	2	0	3
rbg016a	1	1	0	0	0	0	0	0	1	0	3
rbg016b	21	3	0	0	3	0	1	0	17	0	45
rbg017.2	0	0	0	0	0	0	0	0	0	0	0
rbg017	1	0	0	0	2	1	0	0	2	0	6
rbg017a	0	0	0	0	0	0	0	0	0	0	0
rbg019a	0	0	0	0	0	0	0	0	0	0	0
rbg019b	48	0	0	0	0	0	0	0	28	0	76
rbg019c	36	4	90	29	4	4	2	0	9	0	178
rbg019d	2	0	0	0	0	0	0	0	1	0	3
rbg020a	0	0	0	0	0	0	0	0	0	0	0
rbg021.2	3	0	10	0	2	0	0	0	1	0	16
rbg021.3	69	0	78	49	18	1	7	0	14	0	236
rbg021.4	5	0	18	6	19	0	0	0	12	0	60
rbg021.5	8	2	0	0	0	0	1	0	19	0	30
rbg021.6	2	0	0	0	0	0	0	0	5	0	7
rbg021.7	6	8	0	0	0	0	0	0	21	0	35
rbg021.8	7	3	0	0	0	0	0	0	95	0	105
rbg021.9	5	6	0	0	0	0	0	0	193	0	204
rbg021	36	4	90	29	4	4	2	0	9	0	178
rbg027a	9	0	13	0	3	0	2	0	0	0	27
rbg031a	0	0	0	0	0	0	0	0	2	0	2
rbg033a	4	0	0	0	0	0	0	0	5	0	9
rbg034a	0	0	0	0	0	0	0	0	1	0	1
rbg035a.2	13	0	0	0	0	0	0	0	9	0	22
rbg035a	1	0	0	0	0	0	0	0	0	0	1
rbg038a	9000	138	0	0	0	0	0	0	716	0	9854
rbg040a	211	2	0	0	0	0	0	0	39	0	252
rbg041a	17995	121	0	0	0	0	0	0	2569	0	20685
rbg042a	14288	156	0	0	3	1	38	0	2205	0	16691
rbg048a	28631	133	1679	1152	5114	10194	1359	0	1141	0	49403
rbg049a	21246	58	784	416	2147	7568	655	0	829	0	33703
rbg050a	1	0	0	0	0	0	0	0	1	0	2
rbg050b	11632	55	618	347	595	5147	37	0	410	0	18841
rbg050c	30175	480	1150	329	1254	3583	3005	0	4395	0	44371
rbg055a	0	0	0	0	0	0	0	0	1	0	1
rbg067a	0	0	0	0	0	0	0	0	1	0	1
rbg086a	2319	147	4	2	1	503	0	0	122	0	3098
rbg092a	4224	100	0	0	2	1	3	0	445	0	4775
rbg125a	4	0	0	0	0	0	0	0	9	0	13
rbg132.2	465	6	0	0	0	0	0	0	79	0	550
rbg132	2024	11	0	0	4	4	0	0	92	0	2135
rbg152.3	108	0	0	0	0	0	0	0	23	0	131
rbg152	1136	20	0	0	2	1	1	0	167	0	1327
rbg172a	1746	6	0	0	0	0	0	0	458	0	2210
rbg193.2	74	0	0	0	0	0	0	0	47	0	121
rbg193	1407	20	0	0	1	3	1	0	238	0	1670
rbg201a	357	0	0	0	0	0	0	0	19	0	376
rbg233.2	32	0	0	0	0	0	0	0	21	0	53
rbg233	3866	3	0	0	3	10	6	0	482	0	4370

Table 8.5: Number of generated infeasible path constraints

	Init	LP	Improve	Separation	Pricing	Misc	Total
rbg010a	0.0	14.3	14.3	28.6	0.0	42.9	0:00.12
rbg016a	16.7	33.3	8.3	41.7	0.0	0.0	0:00.20
rbg016b	0.4	32.6	13.1	36.0	4.9	13.1	0:08.80
rbg017.2	100.0	0.0	0.0	0.0	0.0	0.0	0:00.03
rbg017	4.1	24.5	16.3	44.9	0.0	10.2	0:00.82
rbg017a	57.1	42.9	0.0	0.0	0.0	0.0	0:00.12
rbg019a	100.0	0.0	0.0	0.0	0.0	0.0	0:00.03
rbg019b	0.1	23.5	11.4	40.2	4.6	20.3	0:54.57
rbg019c	1.1	32.5	7.5	42.6	2.7	13.6	0:08.72
rbg019d	2.2	26.7	13.3	48.9	0.0	8.9	0:00.75
rbg020a	91.7	0.0	0.0	0.0	8.3	0.0	0:00.20
rbg021.2	38.5	38.5	0.0	15.4	7.7	0.0	0:00.22
rbg021.3	0.3	25.8	7.1	50.0	3.3	13.6	0:27.15
rbg021.4	1.4	25.2	4.9	44.7	3.2	20.6	0:05.82
rbg021.5	1.3	27.9	5.8	37.9	5.8	21.4	0:06.63
rbg021.6	7.2	44.6	18.1	22.9	3.6	3.6	0:01.38
rbg021.7	1.9	26.0	27.9	26.7	1.9	15.5	0:04.30
rbg021.8	0.5	26.2	24.1	24.3	3.9	20.9	0:17.40
rbg021.9	0.4	23.5	26.9	22.5	5.7	21.0	0:26.12
rbg021	1.0	34.9	5.3	44.6	3.0	11.2	0:08.75
rbg027a	14.8	28.1	7.4	36.3	5.9	7.4	0:02.25
rbg031a	12.7	24.5	31.4	25.5	2.0	3.9	0:01.70
rbg033a	16.2	18.9	21.6	29.7	0.0	13.5	0:01.85
rbg034a	40.7	25.4	8.5	8.5	11.9	5.1	0:00.98
rbg035a.2	1.0	33.8	23.7	19.1	10.1	12.3	1:04.80
rbg035a	19.1	24.5	28.2	21.8	0.9	5.5	0:01.83
rbg038a	0.0	13.8	5.8	61.2	2.3	16.9	70:32.23
rbg040a	0.1	23.1	5.1	49.8	4.5	17.4	12:31.82
rbg041a	0.0	25.2	10.6	54.5	2.2	7.5	300:01.68
rbg042a	0.0	22.1	16.5	51.3	2.5	7.6	300:01.68
rbg048a	0.0	20.2	3.1	62.1	5.1	9.5	300:01.12
rbg049a	0.0	16.0	3.5	69.8	3.6	7.1	300:01.22
rbg050a	7.3	16.2	44.2	19.6	2.7	10.0	0:18.62
rbg050b	0.0	21.0	2.1	61.5	9.6	5.8	300:01.33
rbg050c	0.0	18.9	5.3	60.2	4.5	11.1	300:01.27
rbg055a	22.1	20.6	16.9	26.6	2.6	11.2	0:06.40
rbg067a	41.2	21.0	17.4	14.3	1.4	4.8	0:05.95
rbg086a	0.0	6.6	21.1	39.3	2.5	30.5	300:01.95
rbg092a	0.0	16.0	9.0	49.2	5.3	20.5	300:01.40
rbg125a	7.8	8.3	17.1	16.4	3.9	46.4	3:49.82
rbg132.2	0.2	13.6	9.4	55.1	4.8	16.9	300:01.42
rbg132	0.1	13.0	8.9	53.5	3.6	20.9	300:01.47
rbg152.3	0.5	42.8	16.9	16.1	9.8	14.0	300:02.90
rbg152	0.2	11.4	12.6	44.0	4.1	27.7	300:03.10
rbg172a	0.3	11.5	7.2	59.7	5.1	16.1	300:02.43
rbg193.2	0.7	29.7	9.2	30.2	8.9	21.3	300:06.55
rbg193	0.4	8.3	6.9	52.0	7.8	24.6	300:03.98
rbg201a	0.5	7.8	6.1	56.8	7.9	20.8	300:06.08
rbg233.2	1.4	28.5	17.4	23.2	8.1	21.4	300:01.28
rbg233	0.7	13.6	11.5	55.8	3.8	14.6	300:07.62

Table 8.6: Percentage of computing time spent in different parts of algorithm

9 Conclusions

Our computational experience indicates that most ATSP-TW instances in the range of up to 50–70 nodes can be solved to optimality via branch&cut codes based on any of the three models. The LPs arising from Models 1 and 3 are larger and, at times, significant numerical instabilities occur in the solution process, despite the fact that our LP solver CPLEX is generally very good at handling numerical difficulties. The reasons for this are model inherent (big M, weak linking constraints) and not due to poor software. Overall the implementation of Model 2 outperformed the two other codes, although for special cases (few time windows active) Model 3 appeared to be more suitable.

The results of our test runs made us conclude that Model 2 is suited best for our particular application of ATSP-TW and we decided to produce a more thorough implementation. This branch&cut code (based on ABACUS and CPLEX 5.0) was tested on real–world and modified real–world instances with sizes of up to about 250 nodes. However, this model has the disadvantage that it can only handle TSP–objective functions but not makespan objectives and the like. For our application, this does not matter, though.

We should also remark here that our computational experience does not indicate that – in contrast to the TSP, say – the polyhedral approach to the ATSP-TW is the unchallenged winning strategy. We believe that dynamic programming and implicit enumeration techniques may outperform our cutting plane method, in particular when the time windows are rather tight. Further research (including very time consuming implementational work) has to show where the relative advantages of the different methodologies lie. Nevertheless, the heuristics and cutting plane algorithms described in this paper were able to solve real–world instances of a particular application (stacker crane optimization) of the asymmetric travelling salesman problem with time windows in a way that is satisfactory for practice.

References

[1] D. Applegate, R. Bixby, V. Chvatal and W. Cook. On the Solution of Traveling Salesman Problem. *Doc.Math.J.DMV* Extra Volume ICM III (1998) 645-656

[2] D. Applegate and W. Cook. A computational study of the job–shop scheduling problem. *ORSA J. on Comp.*, 3:149–156, 1991.

[3] N. Ascheuer. *Hamiltonian Path Problems in the On-line Optimization of Flexible Manufacturing Systems.* PhD Thesis[2], Technische Universität Berlin, 1995.

[4] N. Ascheuer, M. Fischetti, and M. Grötschel. A polyhedral study of the asymmetric travelling salesman problem with time windows. Preprint SC 97-11[3] Konrad-Zuse-Zentrum Berlin, 1997 (to appear in *Networks*).

[5] N. Ascheuer, M. Grötschel, and A. Abdel–Hamid Abdel–Aziz. Order picking in an automatic warehouse: Solving online asymmetric TSPs. *Mathematical Methods of Operations Research*, 49:501–515, 1999.

[2] Available at URL `http://www.zib.de/ZIBbib/Publications/`

[3] Available at URL `http://www.zib.de/ZIBbib/Publications/`

[6] N. Ascheuer, M. Grötschel, S.O. Krumke, and J. Rambau. Combinatorial online optimization. In P. Kall and H.-J. Lüthi, editors, *Operations Research Proceedings 1998*, pages 21–37. Springer, 1999.

[7] N. Ascheuer, M. Jünger, and G. Reinelt. A branch & cut algorithm for the asymmetric Traveling Salesman Problem with precedence constraints. Preprint SC 97-70[4], Konrad-Zuse-Zentrum Berlin, 1997 (to appear in *Computational Optimization and Applications*).

[8] E.K. Baker. An exact algorithm for the time–constrained traveling salesman problem. *Operations Research*, 31(5):938–945, 1983.

[9] E. Balas. On the facial structure of scheduling polyhedra. *Mathematical Programming Study*, 24:179–218, 1985.

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

[11] E. Balas, M. Fischetti, and W. Pulleyblank. The precedence constrained asymmetric traveling salesman polytope. *Math. Prog.*, 68:241–265, 1995.

[12] E. Balas and N. Simonetti. Linear time dynamic programming algorithms for some classes of restricted tsp's. Technical Report MSRR No.617, Carnegie Mellon University, Pittsburgh, USA, 1996.

[13] M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, editors. *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*. Elsevier Sci. B.V., Amsterdam, 1995.

[14] L. Bianco, A. Mingozzi, and S. Ricciardelli. Dynamic programming strategies and reduction techniques for the travelling salesman problem with time windows and precedence constraints. *Operations Reserach*, 45(3):365–377, 1997.

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

[16] M. Desrochers, J. Desrosiers, and M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Reserach*, 40(2):342–354, 1992.

[17] M. Desrochers and G. Laporte. Improvements and extensions to the Miller–Tucker–Zemlin subtour elimination constraints. *Operations Research Letters*, 10(1):27–36, 1991.

[18] M. Desrochers, J.K. Lenstra, M.W.P. Savelsbergh, and F. Soumis. Vehicle routing with time windows: Optimization and approximation. In B.L. Golden and A.A. Assad, editors, *Vehicle routing: Methods and studies*, pages 65–84. North–Holland, 1988.

[19] J. Desrosiers, Y. Dumas, M.M. Solomon, and F. Soumis. *Time Constrained Routing and Scheduling*, chapter 2, pages 35–139. Volume 8 of Ball et al. [13], 1995.

[20] Y. Dumas, J. Desrosiers, E. Gelinas, and M.M. Solomon. An optimal algorithm for the traveling salesman problem with time windows. *Operations Research*, 43(2):367–371, 1995.

[4] Available at URL `http://www.zib.de/ZIBbib/Publications/`

[21] M. Dyer and L.A. Wolsey. Formulating the single machine sequencing problem with release dates as a mixed integer program. *Disc. Applied Math.*, 26:255–270, 1990.

[22] M. Fischetti. Facets of the asymmetric traveling salesman polytope. *Mathematics of Operations Research*, 16:42–56, 1991.

[23] M. Fischetti and P. Toth. A polyhedral approach to the asymmetric traveling salesman problem. *Management Science*, 43(11):1520–1536, 1997.

[24] M.R. Garey and D.S. Johnson. Two–processor scheduling with start–times and deadlines. *SIAM Journal on Computing*, 6:416–426, 1977.

[25] M. Grötschel. *Polyedrische Charakterisierungen kombinatorischer Optimierungsprobleme*. Hain, Meisenheim am Glan, 1977.

[26] M. Grötschel and M. Padberg. Polyhedral computations. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, editors, *The Traveling Salesman Problem*. John Wiley & Sons, 1985.

[27] M. Grötschel and M. Padberg. Polyhedral theory. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, editors, *The Traveling Salesman Problem*. John Wiley & Sons, 1985.

[28] M. Jünger, G. Reinelt, and G. Rinaldi. The traveling salesman problem. In M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, editors, *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, chapter 4, pages 225–330. North Holland, 1995.

[29] M. Jünger, G. Reinelt, and S. Thienel. Provably good solutions for the traveling salesman problem. *Zeitschrift für Operations Research*, 40(2):183–217, 1994.

[30] M. Jünger and S. Thienel. Introduction to ABACUS – A Branch And CUt System. Technical report, Institut für Informatik, Universität zu Köln, Technical Report No. 97.263, 1997. See on-line documentation under URL
`http://www.informatik.uni-koeln.de/ls_juenger/projects/abacus.html`.

[31] M. Jünger and S. Thienel. Introduction to ABACUS - a branch and cut system. *Operations Research Letters*, 22:83–95, 1998.

[32] F. Maffioli and A. Sciomachen. A mixed-integer model for solving ordering problems with side constraints. Technical Report Research report, University of Genoa, Italy, 1993.

[33] C.E. Miller, A.W. Tucker, and R.A. Zemlin. Integer programming formulations and traveling salesman problems. *J. Assoc. Comput. Mach.*, 7:326–329, 1960.

[34] I. Or. *Traveling Salesman–Type Combinatorial Problems and their relation to the logistics of regional blood banking*. PhD Thesis, Dept. of Industrial Engineering and Management Science, Northwestern University, Evanston, 1976.

[35] M. Padberg and G. Rinaldi. An efficient algorithm for the minimum capacity cut problem. *Mathematical Programming*, 47:19–36, 1990.

[36] M. Padberg and G. Rinaldi. A Branch and Cut Algorithm for the Resolution of Large–Scale Symmetric Traveling Salesman Problems. *SIAM Review*, 33:60–100, 1991.

[37] M. Savelsbergh, 1994. Personal communication (1994) . School of Industrial and System Engineering, Georgia Institute of Technology, Atlanta, USA.

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

[39] M.W.P. Savelsbergh. The vehicle routing problem with time windows: minimizing route duration. COSOR Memorandum 91–03, Eindhoven University of Technology, 1991.

[40] N. Simonetti, 1997. Personal communication (1997).

[41] M.M. Solomon, E.K. Baker, and J.R. Schaffer. Vehicle routing and scheduling problems with time window constraints: Efficient implemantations of solution improvement procedures. In B.L. Golden and A.A. Assad, editors, *Vehicle routing: Methods and studies*, pages 85–105. North–Holland, 1988.

[42] S. Thienel. *ABACUS A Branch-And-CUt System.* PhD thesis, Univ. zu Köln, 1995.

[43] J. Tsitsiklis. Special cases of traveling salesman and repairman problems with time windows. *Networks*, 22:263–282, 1992.

[44] C.A. van Eijl. A polyhedral approach to the delivery man problem. Technical Report 95–19, Department of Mathematics and Computer Science, Eindhoven University of Technology, 1995.