Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

NORBERT ASCHEUER MICHAEL JÜNGER
GERHARD REINELT

A Branch & Cut Algorithm for the Asymmetric Traveling Salesman Problem with Precedence Constraints

Preprint SC 97-70 (December 1997; revised January 1999)

A Branch & Cut Algorithm for the Asymmetric Traveling Salesman Problem with Precedence Constraints

Norbert Ascheuer * Michael Jünger † Gerhard Reinelt ‡

Berlin, December 1997; revised January 1999

Abstract

In this article we consider a variant of the classical asymmetric traveling salesman problem (ATSP), namely the ATSP in which precedence constraints require that certain nodes must precede certain other nodes in any feasible directed tour. This problem occurs as a basic model in scheduling and routing and has a wide range of applications varying from helicopter routing [25], sequencing in flexible manufacturing [3, 4], to stacker crane routing in an automatic storage system [2].

We give an integer programming model and summarize known classes of valid inequalities. We describe in detail the implementation of a branch&cut-algorithm and give computational results on real–world instances and benchmark problems from TSPLIB. The results we achieve indicate that our implementation outperforms other implementations found in the literature. Real world instances with more than 200 nodes can be solved to optimality within a few minutes of CPU-time.

As a side product we obtain a branch&cut-algorithm for the ATSP. All instances in TSPLIB can be solved to optimality in a reasonable amount of computation time.

Keywords: Asymmetric traveling salesman problem, precedence constraints, branch&cut

1 Introduction

The *asymmetric traveling salesman problem with precedence constraints*, also called *sequential ordering problem*, can be phrased in graph theoretical terminology as follows. We are given a complete directed graph $D_n = (V, A_n)$ on n nodes, and cost-coefficients $c_{ij} \in \mathbb{N}, c_{ij} \geq 0$, associated with each arc $(i, j) \in A_n$. We assume without loss of generality that a fixed starting node for each tour, say node 1, is given. Furthermore, we are given an additional precedence digraph $P = (V, R)$ that is defined on the same node set V as D_n. An arc $(i, j) \in R$, $i, j \neq 1$, represents a precedence relationship, i.e., i has to precede j in every feasible tour. This will also be denoted by $i \prec j$. Obviously, the precedence digraph P must be acyclic, otherwise no feasible solution exists. Moreover, it can be assumed to be transitively closed, because if $i \prec j$ and $j \prec k$ we can conclude that $i \prec k$.

*Konrad–Zuse–Zentrum für Informationstechnik Berlin, Takustr. 7, D–14195 Berlin Dahlem

†Institut für Informatik der Universität zu Köln, Pohligstr. 1, D–50969 Köln

‡Institut für Angewandte Mathematik, Universität Heidelberg, Im Neuenheimer Feld 293, D–69120 Heidelberg

A tour that satisfies the given precedence relationships will be called *feasible*. The problem is to find a feasible tour with minimal total cost. In the literature this problem is also known as the *sequential ordering problem (SOP)*. Obviously, the SOP reduces to the "pure" asymmetric traveling salesman problem (ATSP) in the case that the precedence digraph $P = (V, R)$ has empty arc set R. Thus, the ATSP is a special case of the SOP. As the ATSP is an $\mathcal{NP}$-hard problem the same holds for the SOP.

There is a great variety of real–world problems that can be modeled as a SOP. They occur, e.g., in routing applications where a pick-up has to precede the delivery [26], in the on-line routing of a stacker crane in an automatic storage system [2] or in scheduling applications where a certain job has to be completed before other jobs can start [4].

Although the traveling salesman problem is one of the most investigated problems in combinatorial optimization, not much attention has been paid to the precedence constrained version. The sequential ordering problem (SOP), stated in a slightly different form, seems to have been mentioned first by Escudero [9, 10]. The aim of his investigations was to design heuristics to be implemented in a production planning system that perform well in practice with respect to solution quality guarantee and running time.

Ascheuer [1] and Ascheuer, Escudero, Grötschel, and Stoer [3, 4] describe a cutting-plane approach to obtain lower bounds on the value of the optimal solution. These lower bounds were used to check the quality of the solutions constructed by the heuristics developed by Escudero. The authors compare three different models and give computational results on real–life data of IBM that was provided by Escudero. The model that will be presented in Section 2 turned out to give satisfactory bounds and to be superior to the other models from a computational point of view. Escudero, Guignard, and Malik [11] strengthened the model by introducing new classes of valid inequalities and describe a Lagrangean relax-and-cut approach that for some instances obtained better lower bounds than the ones given by Ascheuer et al.

Around the same time a similar model was developed independently by Balas, Pulleyblank, and Timlin [25, 26]. They considered the symmetric case and the aim of their research was to design heuristics to schedule helicopters that have to visit offshore oil-platforms in a certain order. It was desired to find a route for each daily set of stops that satisfies all the requirements (precedence constraints, helicopter capacity) and minimizes the total distance flown.

For the TSP and ATSP polyhedral approaches (in particular branch&cut-algorithms) are known to be the appropriate method to solve problem instances to optimality (see [13, 17], among others). For the *sequential ordering polytope*, i.e., the convex hull of the incidence vectors of all feasible solutions, first polyhedral investigations have been carried out. We will summarize some of the inequalities presented in Balas, Fischetti, and Pulleyblank [7] and Ascheuer [2] in Section 3. Although there exist partial results on the structure of this polytope, to the best of our knowledge there is no exact algorithm published that exploits these polyhedral results.

Branch&cut-algorithms simultaneously calculate series of increasing lower and decreasing upper bounds. In case that they coincide, the optimality of the feasible solution has been proven. Even if this is not the case the bounds allow to give a quality guarantee on the best solution, stating that this solution is no more than a certain percentage away from an optimal one. This is in contrast to pure heuristic approaches. Moreover, the algorithm may be stopped if a prespecified quality is reached.

In the sequel, we use the following terminology. Given a digraph $D = (V, A)$, variables x_{ij} for

all $(i,j) \in A$, $F \subseteq A$ and $W \subseteq V$, we set $x(F) := \sum_{(i,j)\in F} x_{ij}$ and $A(W) := \{(i,j) \in A \mid i,j \in W\}$. If $j \in V$, then $\delta^-(j) := \{(i,j) \in A,\ \ i \in V \setminus \{j\}\}$ and $\delta^+(j) := \{(j,k) \in A,\ \ k \in V \setminus \{j\}\}$. For $U, W \subseteq V, U \cap W = \emptyset$, we set $(U : W) := \{(i,j) \in A \mid i \in U, j \in W\}$. To simplify notation, we write $(j : W)$ instead of $(\{j\} : W)$. Given a (transitively closed) precedence digraph $P = (V,R)$, the set $\pi(j)$ of *predecessors* of $j \in V$ and the set $\sigma(i)$ of *successors* of i are defined by

$$\begin{aligned} \pi(j) &:= \{i \in V \mid (i,j) \in R\}, \\ \sigma(i) &:= \{j \in V \mid (i,j) \in R\}. \end{aligned}$$

If there exists a node $i \in V \setminus \{1\}$ such that $|\pi(i)| + |\sigma(i)| = n - 2$ we know that node i is *fixed* to a certain position in the tour and thus the problem instance can be split into two instances. If this is the case, we say that the instance is *decomposable*.

The paper is organized as follows. In Section 2 we present an integer programming model for the SOP and introduce the sequential ordering polytope. Section 3 contains a summary of known classes of valid inequalities that will be used in the branch&cut-algorithm. Section 4 describes the implementation details of the branch&cut-algorithm. Section 5 outlines the computational experiments. The paper closes with some concluding remarks.

2 Modeling

Suppose we are given a complete digraph $D = (V, A_n)$ and a precedence digraph $P = (V,R)$. For each arc $(i,j) \in A_n$ we introduce a binary variable $x_{ij} \in \{0,1\}$ with the interpretation that

$$x_{ij} = \begin{cases} 1, & \text{if } (i,j) \in A_n \text{ is in the tour,} \\ 0, & \text{otherwise.} \end{cases}$$

Immediately some simple reductions can be performed.

(1) Lemma.
Given $D_n = (V, A_n)$ and $P = (V,R)$, then
(i) $x_{1j} = 0$ for all $(i,j) \in R$
(ii) $x_{i1} = 0$ for all $(i,j) \in R$
(iii) $x_{ji} = 0$ for all $(i,j) \in R$
(iv) $x_{ik} = 0$ for all $(i,k) \in R$ such that $(i,j) \in R$ and $(j,k) \in R$.

Proof. Obvious. □

The variable set (resp. arc set) can be reduced by fixing these variables to 0 (resp. deleting the arcs). To state this more formally we let

$$\begin{aligned} R_1 &= \{(1,j),(i,1) \in A_n \mid (i,j) \in R\}, \\ R_2 &= \{(j,i) \in A_n \mid (i,j) \in R\}, \\ R_3 &= \{(i,k) \in A_n \mid (i,j) \in R \text{ and } (j,k) \in R\}. \end{aligned}$$

By setting

$$A = A_n \setminus (R_1 \cup R_2 \cup R_3)$$

we end up with the digraph $D = (V,A)$ in which we look for a tour of minimum length. We only associate variables with the *feasible arc set* A. In [7] it was proven that this lemma yields a complete characterization of all variables that can a priori be fixed to zero.

The linear 0/1-model can be stated as follows (see [4]) :

(2) Linear 0/1-Programming Formulation.

$$
\begin{array}{lll}
 & \min c^T x & \\
\text{s. t. } (1) & x(\delta^-(i)) = 1 & \forall\, i \in V \\
(2) & x(\delta^+(i)) = 1 & \forall\, i \in V \\
(3) & x(A(W)) \leq |W| - 1 & \forall\, W \subset V,\ 2 \leq |W| \\
(4) & x(j:W) + x(A(W)) + X(W:i) \leq |W| & \forall\, (i,j) \in R \text{ and} \\
 & & \forall\, W \subseteq V \setminus \{1,i,j\},\ W \neq \emptyset \\
(5) & x_{ij} \in \{0,1\} & \forall\, (i,j) \in A
\end{array}
$$

(1)–(3),(5) is the standard formulation for the asymmetric traveling salesman problem. Inequalities (3) are called *subtour elimination constraints*, the inequality system (4) assures that all precedences are satisfied (*precedence forcing constraints*).

It is readily checked that every feasible solution of (1)–(5) is the incidence vector of a feasible tour and vice-versa. Although this model yields an integer programming formulation, it should be mentioned that inequalities (4) can be strengthened by adding either $x(W:j)$ or $x(i:W)$ to the left hand side and that several classes of inequalities are known that strictly dominate these strengthened precedence forcing inequalities (see [7]). These inequalities, that are used in our implementation, will be described in the following section.

3 Classes of valid inequalities

In the following we summarize classes of inequalities that we use in our implementation of the branch&cut-algorithm and that are known to be valid for the *sequential ordering polytope*

$$SOP(n,P) = \text{conv}\{x \in \mathbb{R}^A \mid x \text{ is a feasible tour with respect to } P\}.$$

The study of the structure of this polytope is closely related to the study of the asymmetric traveling salesman polytope P_T^n, as $SOP(n,P) \subseteq P_T^n$ holds. In the literature several classes of valid and facet defining inequalities are known for P_T^n (see [12, 15, 16], among others). Obviously, these inequalities are valid for $SOP(n,P)$ as well. In case that precedences between the nodes in V are involved, they can be strengthened in several ways. The polyhedral study of the sequential ordering polytope turned out to be complicated [2, 7]. So far, no general formula for the dimension of $SOP(n,P)$ has been proven. Therefore, the classes of inequalities have only been proven to be valid for $SOP(n,P)$. Whether and under which conditions they are facet defining still remains open.

The classes of inequalities that we use include:

- "Pure" ATSP-inequalities (D_k-inequalities, SD-inequalities),
- strengthened versions of ATSP-inequalities (2-matching inequalities, T_k-inequalities, π-inequalities, etc.),
- classes of inequalities introduced for the SOP.

3.1 D_k-inequalities

The D_k-inequalities, introduced by Grötschel and Padberg [15, 16], are obtained by sequentially lifting the cycle inequality $\sum_{(i,j)\in C} x_{ij} \leq k-1$ for the cycle $C = \{(i_1, i_2), \ldots, (i_k, i_1)\}$. Depending on the order in which the variables are lifted one obtains different classes of inequalities, two of them called D_k^--inequalities

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

and D_k^+-inequalities

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

D_k-inequalities have been proven to be facet-defining for P_T^n by Fischetti [12]. Strengthenings of the D_k-inequalities for the precedence constrained ATSP are known for small k, e.g., $k = 3$ (see [2]).

3.2 SD-inequalities

SD-inequalities have been introduced by Balas and Fischetti [6] and can be described as follows. Given a *handle* $H \subset V$, disjoint *teeths* $T_1, ..., T_t$, t odd, such that $|T_i \cap H| = 1$ and $|T_i \setminus H| = 1$, and (possibly empty) disjoint node sets S and D, $(S \cup D) \subset V \setminus (H \cup T_1 \cup ... \cup T_t)$, such that $|S| + |D| = t$ is odd, the SD-inequalities have the form

$$x(S \cup H : D \cup H) + \sum_{i=1}^{t} x(A(T_i)) \leq |H| + \frac{|S| + |D| + t - 1}{2}. \tag{5}$$

SD-inequalities may be seen as a generalization of 2-matching inequalities to which a source- and destination node set is attached.

3.3 Predecessor / Successor inequalities

Balas, Fischetti, and Pulleyblank [7] introduced classes of inequalities that can be seen as strengthenings of the subtour elimination inequality (written in the cut form)

$$x(\delta^+(W)) \geq 1 \text{ for all } W \subset V.$$

Let $S \subseteq V \setminus \{1\}, \overline{S} := V \setminus S$, then the *predecessor inequality (π-inequality)*

$$x(S \setminus \pi(S) : \overline{S} \setminus \pi(S)) \geq 1 \tag{6}$$

and the *successor inequality (σ-inequality)*

$$x(\overline{S} \setminus \sigma(S) : S \setminus \sigma(S)) \geq 1 \tag{7}$$

are valid with respect to $SOP(n, P)$ [7]. For given $j, k \in V \setminus \{1\}$ such that $\pi(j) \neq \emptyset, \sigma(k) \neq \emptyset$ and any $S \subset V$ such that $k, j \in S$, the inequalities

$$x(S \setminus \pi(j) : \overline{S} \setminus \pi(j)) \geq 1 \tag{8}$$

$$x(\overline{S} \setminus \sigma(k) : S \setminus \sigma(k)) \geq 1 \tag{9}$$

are called weak π- and weak σ-inequality.

Furthermore, let $X, Y \subseteq V \setminus \{1\}$, such that $i \prec j$ for all pairs $i \in X, j \in Y$, $W := \{1\} \cup \pi(X) \cup \sigma(Y)$. Then for all $S \subset V$, such that $X \subset S, Y \subset \overline{S}$

$$x(S \setminus W : \overline{S} \setminus W) \geq 1 \tag{10}$$

is called a *predecessor-successor inequality* or *(π, σ)-inequality* and is valid with respect to $SOP(n, P)$. For $X = \{i\}$ and $Y = \{j\}$ with $i \prec j$ and $W_{ij} := \{1\} \cup \pi(i) \cup \sigma(j)$

$$x(S \setminus W_{ij} : \bar{S} \setminus W_{ij}) \geq 1 \tag{11}$$

is called a *weak (π, σ)-inequality*.

Balas, Fischetti and Pulleyblank proved that the π- and σ-inequalities are facet defining under certain conditions (see [7]).

3.4 Precedence cycle breaking inequalities

Let $S_1, .., S_m \subseteq V \setminus \{1\}, m \geq 2$, be disjoint node sets such that $\sigma(S_i) \cap S_{i+1} \neq \emptyset$ with $S_{m+1} = S_1$. Then the *precedence cycle breaking inequality* (or *pcb-inequality*)

$$\sum_{i=1}^{m} x(A(S_i)) \leq \sum_{i=1}^{m} |S_i| - m - 1 \tag{12}$$

is valid for $SOP(n, P)$ [7].

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

$$x(A(S_1)) \leq |S_1| - 2 \tag{13}$$

and is as well a strengthened version of a subtour elimination inequality, in which the right hand side is reduced by one. This inequality will be called a *simple pcb-inequality*.

Let $S_1, S_2, S_3 \subset V \setminus \{1\}$ be disjoint node sets, with $\sigma(S_1) \cap S_2 \neq \emptyset$ and $\sigma(S_2) \cap S_3 \neq \emptyset$. The following inequalities are valid with respect to $SOP(n, P)$ [7]:

$$\sum_{i=1}^{2} x(A(S_i)) + x(S_2 : S_1) \leq |S_2| + |S_1| - 2, \tag{14}$$

$$\sum_{i=1}^{3} x(A(S_i)) + x(S_1 : S_3) \leq |S_1| + |S_2| + |S_3| - 3. \tag{15}$$

Roughly speaking, inequalities (14) and (15) may be interpreted as being obtained from the trivial inequalities $x_{ij} \leq 0$ for all infeasible arcs $(i, j) \in A_n$ (characterized by Lemma 1) by a clique-lifting procedure akin to that described in [6], in which each node v_i is replaced by a clique S_i of indistinguishable "clones".

3.5 Strengthened T_k-inequalities

Let $W \subset V, 2 \leq |W| = k \leq n - 2$, $w \in W$, $p, q \in V \setminus W$. The T_k-inequalities

$$x(A(W)) + x_{pw} + x_{pq} + x_{wq} \leq k$$

have been introduced by Grötschel and Padberg and have been proven to be facet defining for P_T^n for most k and n (see, e.g., [15, 16]).

Like the subtour elimination inequalities, these inequalities can be strengthened as precedences get involved. As they are more "complicated" than the subtour elimination inequalities, the resulting strengthened versions do not have such a nice form as the π-inequalities, etc. All strengthened T_k-inequalities we consider have been introduced by Ascheuer [2]. In the sequel let

$$x(A(T_k)) := x(A(W)) + x_{pw} + x_{pq} + x_{wq}$$

denote the sum of the left hand side of the T_k-inequality. Furthermore, let $\widetilde{W} = W \setminus \{w\}$. Then the inequalities

$$\begin{aligned} x(A(T_k)) + x(\overline{W} \cap \sigma(\widetilde{W}) : W) + x(\overline{W} \setminus \sigma(\widetilde{W}) : W \cap \sigma(\widetilde{W})) &\leq k \qquad (16) \\ x(A(T_k)) + x(W : \overline{W} \cap \pi(\widetilde{W})) + x(W \cap \pi(\widetilde{W}) : \overline{W} \setminus \pi(\widetilde{W})) &\leq k \qquad (17) \end{aligned}$$

are valid with respect to $SOP(n, P)$. Further valid inequalities are known for the case that special structures in the precedences occur: Let $i, k \in \widetilde{W}$, $j \notin W \cup \{p, q\}$. Then the following inequalities are valid for $SOP(n, P)$:

$$\begin{aligned} x(A(T_k)) &\leq k-1 \quad , \quad \text{if } i \prec j \prec k, \qquad (18) \\ x(A(T_k)) + x(p : \widetilde{W}) + x_{wp} &\leq k-1 \quad , \quad \text{if } i \prec q \prec w, \qquad (19) \\ x(A(T_k)) + x(\widetilde{W} : q) + x_{qw} &\leq k-1 \quad , \quad \text{if } w \prec p \prec k \qquad (20) \end{aligned}$$

3.6 Strengthened 2-matching inequalities

2-matching inequalities are widely used in implementations of branch&cut-algorithms for the TSP, because exact as well as efficient heuristic separation procedures exist. As the 2-matching inequalities are valid for $SOP(n, P)$, the separation routines can be applied to our problem as well. As done for the T_k-inequalities the 2-matching inequalities can be strengthened in case that precedences get involved.

Let $H, T_1, T_2, ..., T_k \subset V, k \geq 1$ be vertex sets satisfying

$$\begin{array}{ll} |H \cap T_i| = 1 & \text{for } i = 1, ..., k, \\ |T_i \setminus H| = 1 & \text{for } i = 1, ..., k, \\ T_i \cap T_j = \emptyset & \text{for } 1 \leq i < j \leq k, \\ k \geq 3 \text{ and odd}, & \text{or } k = 1 \text{ and } |H| \geq 4. \end{array}$$

The 2-matching inequality is given by

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

To simplify notation further, let $T = \bigcup_{i=1}^{k} T_i$, $S = H \cup T$. Let $\{s_i\} := H \cap T_i$, $\{t_i\} := T_i \setminus H$, and $x(2M) := x(A(H)) + \sum_{i=1}^{k} x(A(T_i))$.

The following inequalities are valid with respect to $SOP(n, P)$ (see [2]):

$$x(2M) + \sum_{i=1}^{k} x(\overline{S} \cap \sigma(t_i) : s_i) + \sum_{i=1}^{k} x(s_i : \overline{S} \cap \pi(t_i)) \leq |H| + \frac{k-1}{2} \qquad (21)$$

$$x(2M) + \sum_{i=1}^{k} x((H \setminus T) \cap \sigma(s_i) : t_i) + \sum_{i=1}^{k} x(t_i : (H \setminus T) \cap \pi(s_i)) \leq |H| + \frac{k-1}{2} \quad (22)$$

$$x(2M) + \sum_{i=1}^{k} x(H \setminus s_i : t_i \cap \sigma(T_i)) \leq |H| + \frac{k-1}{2} \quad (23)$$

$$x(2M) + \sum_{i=1}^{k} \sum_{\substack{j=1 \\ i \neq j}}^{k} x(s_i \cap \pi(T_i) : t_j) \leq |H| + \frac{k-1}{2} \quad (24)$$

$$x(2M) + \sum_{i=1}^{k} x(t_i \cap \pi(T_i) : H) \leq |H| + \frac{k-1}{2} \quad (25)$$

$$x(2M) + \sum_{i=1}^{k} \sum_{\substack{j=1 \\ i \neq j}}^{k} x(t_j : s_i \cap \sigma(T_i)) \leq |H| + \frac{k-1}{2} \quad (26)$$

If there exist $m := \frac{k+1}{2}$ triples $(u_i, v_i, w_i),\ i = 1, ..., m$, such that

$$\begin{array}{ll} \{u_i, w_i\} \in H \cap T, & \\ v_i \in \bar{H} \setminus T, & \\ u_i \prec v_i \prec w_i & \text{for } i = 1, ..., m, \\ v_i \neq v_j & \text{for } i, j = 1, ..., m, i \neq j, \\ w_i = u_{i+1} & \text{for } i = 1, ..., m-1, \end{array}$$

then the inequality

$$x(A(H)) + \sum_{i=1}^{k} x(A(T_i)) \leq |H| + \frac{k-1}{2} - 1 \quad (27)$$

is valid with respect to $SOP(n, P)$ [2].

Inequalities (21)–(26) are strengthenings of the standard 2-matching inequality in the sense that additional variables are added on the left hand side. For inequality (27) it is possible to reduce the right hand side by 1, if a certain structure in the precedences is present.

4 The Branch&Cut-Algorithm

One of the main features of branch&cut-algorithms is that they simultaneously calculate upper and lower bounds on the value of the optimal solution. The upper bounds are provided by heuristic algorithms, whereas lower bound calculations are done by means of a cutting plane algorithm. In case that upper and lower bounds coincide, the optimality of the feasible solution has been proven. In this section we describe in detail the procedures to calculate these bounds.

For an efficient implementation several other details besides upper and lower bound calculations have to be clarified. Since the initial version of our computer code was implemented a few years ago and continuously updated and refined since then, our implementation is based

on the branch&cut software framework that is the predecessor of the ABACUS system [19, 20]. The same framework was used in the branch&cut-code for the TSP that is described in [18]. It consists of a library of `C`-functions that has in the meantime been replaced by the object-oriented `C++`-based ABACUS system. The article [18] contains a detailed description of the general branch&cut-framework. Here we concentrate only on those parts that differ from the description given there and that we consider important.

4.1 Upper bound calculations

We calculate feasible solutions at two points of the algorithm: in the initialization phase and after each solution of a linear program (LP). It turns out that for some instances it is very hard to find good solutions using the initial heuristics. Especially for these instances it is necessary to improve the solution by exploiting the information on the structure of an optimal solution that is contained in the LP-solution. The computational experiments show that very seldomly an optimal solution is found by the initial heuristics.

Initial feasible solution

We implemented a number of modified versions of known heuristics for the TSP. Among them are several construction heuristics (savings, greedy, nearest neighbor, farthest insertion, best insertion, etc.) and additional improvement heuristics (2-node-exchange, 2-Opt, 3-Opt, etc.). We performed an extensive computational comparison of the heuristics and our experience was that none of the heuristics performed well on every instance. For any combination of construction and improvement heuristic we found several problem instances on which they yield rather poor results [5]. Better feasible solutions are obtained by the LP-exploitation heuristic.

Therefore, we decided not to spend much time in constructing an initial feasible solution and we just run a nearest feasible neighbor heuristic. Roughly speaking we start with a partial feasible path T containing only an arc $(1, i)$. Let j be the current last node in T. We successively expand T by a node k such that all predecessors of k are already contained in T and arc (j, k) has minimal cost among all feasible arcs (j, l). We start this construction with each feasible arc $(1, i) \in A$.

Afterwards, the best sequence is passed to a modified 3-opt heuristic in which infeasible changes of the incumbent best solution are rejected.

LP-exploitation heuristic

After each solution of an LP we try to exploit the information contained in the optimal LP-solution, say $\bar{x}$, to find a better solution. Very seldomly, $\bar{x}$ is the incidence vector of a feasible tour. But even a fractional solution contains information on the structure of "good" feasible solutions, as it contains variables with value 1 or close to 1.

First, we check if the current LP-solution is the incidence vector of a feasible tour. If this is the case, we proceed with a pricing step.

If not, we apply a "greedy-like" heuristic in which arcs are sorted according to their value in the current LP-solution. Arcs corresponding to nonactive variables are added to the list with value 0. This list is scanned and arcs are chosen to be part of the sequence if the partial paths remain "feasible". Checking if a certain selection is feasible is more complicated than for the ATSP, because it is not only necessary to avoid subtours, but also to assure that the

given precedences are not violated. (See [5] for a more detailed description.) This heuristic turns out to give good results for dense precedence structures, i.e., $|R| > \alpha \cdot |V|$ for $\alpha \sim 1.5$.

For sparse precedences better results can be achieved with the following approach that takes the cost-coefficients into account as well. Roughly speaking, we modify the original cost-coefficients and run heuristics with the modified cost-matrix $C' = (c'_{ij})_{i,j\in\{1,\ldots,n\}}$ where $c'_{ij} := (1.0 - \overline{x}_{ij}) \cdot c_{ij}$. For nonactive variables x_{lk} we assume $\overline{x}_{lk} = 0$. The goal is to make arcs, whose corresponding variable has a high value, attractive for the heuristic algorithms. After constructing the modified cost-matrix C' we run a construction heuristic with C'. In order to avoid that the same sequence is generated in several subsequent iterations, we vary the heuristic that is called. After the solution of the k-th linear program, we call the

$$\left.\begin{array}{l}\text{list insertion heuristic}\\ \text{random insertion heuristic}\\ \text{nearest feasible neighbor heuristic}\end{array}\right\} \text{ if } k \bmod 3 = \left\{\begin{array}{l}0,\\ 1,\\ 2.\end{array}\right.$$

The constructed feasible tour is the starting point for subsequent improvement heuristics. Since the improvement heuristics can be rather time consuming, we avoid calling them more than once with the same input sequence by using a hash table. The insertion key is the value of the input solution and the index of the improvement heuristic. If the heuristic was already called with a sequence of this value, it is skipped. Notice that different sequences may have the same value and we may miss a solution that leads to a better sequence. Nevertheless, this strategy led to a dramatic reduction in the computation time needed for this part.

Furthermore, we avoid calling the improvement heuristic if the value $\dot{c}$ of the input sequence is "too far away" from the value $\dot{c}^b$ of the incumbent best known solution. In the current implementation we run the improvement part only if $\frac{c^i}{c^b} \leq 2.5$ holds.

Whenever possible, any of the following improvement heuristics is called (with the original cost-matrix C):

- Node reinsertion heuristic
- Two node exchange heuristic
- 3-Opt heuristic

When a better solution is found, the current best solution and the global upper bound are updated, and nonactive variables, whose corresponding arc is in this solution, are added to the sparse digraph and to the LP.

4.2 Separation procedures.

The core of any cutting-plane algorithm is the generation of violated valid inequalities that are added as cutting planes to the current linear program. In the following, we briefly outline how violation of the inequalities described in Section 3 is determined.

An ATSP-inequality $ax \leq \alpha$ is called symmetric if $a_{ij} = a_{ji}$ for all $(i,j) \in A$ holds. Given a valid TSP-inequality $by \leq \beta$, one can derive a symmetric ATSP-inequality $ax \leq \alpha$ by replacing y_e by $x_{ij} + x_{ji}$ and setting $\alpha = \beta, a_{ij} = a_{ji} = b_e$ for all $e = ij \in E$. Conversely, every symmetric ATSP-inequality corresponds to a TSP-inequality.

As a consequence, separation routines implemented for the TSP can be used without any modification for the ATSP and SOP. Suppose we are given a fractional point $\overline{x}$. We

construct the undirected counterpart $\overline{y}$ by setting $\overline{y}_e = \overline{x}_{ij} + \overline{x}_{ji}$ for all $e \in E, e = ij$ and then apply the TSP-separation routine to $\overline{y}$. If a violated inequality is found, the TSP-inequality is retransformed into the corresponding ATSP-version. Both inequalities have the same amount of violation. In our implementation we apply this procedure to the subtour elimination inequalities and to the 2-matching inequalities.

Inequality-Pool. During the run of the algorithm we maintain a *pool of active and non-active 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 an inequality that is only stored in the pool is 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 an inequality 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 regenerate 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. Since not all pool inequalities are active, i.e., are considered in the actual LP, this can be done by "checking" all nonactive pool-inequalities. In case that the pool gets too large, inequalities that have not been reactivated for a long time are removed from the pool.

In contrast to implementations of branch&cut-algorithms for the TSP we separate all types of inequalities from the pool. This is done in order to avoid double entries for certain inequalities (e.g., π-inequalities generated by the separation routine for subtour elimination inequalities).

Subtour Elimination Inequalities. We apply the exact separation algorithm as described by Padberg and Rinaldi [21]. Whenever a violated subtour elimination inequality is found, we check if it can be strengthened as a π-, σ-, (π, σ)-inequality, or as a simple pcb-inequality. If this is the case, we add any of the tightened inequalities.

π-, σ-, (π, σ)-inequalities. Balas et al. [7] describe exact separation procedures for weak π-, σ-, (π, σ)-inequalities. The separation routine for weak π-inequalities roughly works as follows. Suppose we are given a fractional point x^*. We set up a capacitated LP-solution digraph $D^* = (V, A^*)$ with $(i, j) \in A^*$, if $x^*_{ij} > 0$. To each $(i, j) \in A^*$ we associate a capacity $c^*_{ij} = x^*_{ij}$. For all $j \in V$ such that $|\pi(j)| > 0$ we apply the following procedure. We construct a digraph $\tilde{D} = (\tilde{V}, \tilde{A})$ from D^* by deleting all nodes in $\pi(j)$. If we do not succeed to send one unit of flow from j to 1 in $\tilde{D}$, the minimum capacity cut in $\tilde{D}$ separating j from 1 defines a violated weak π-inequality.

The bottleneck in this procedure are the maxflow calculations that have to be called at most n times. In order to avoid them as much as possible, we check in a first step the 1-paths in D^* for possible violations. If this is not successful, we shrink the 1-paths to single nodes and delete all arcs that will not occur in a maximum flow from j to 1, i.e., $(i, j), (1, i) \in A$ for all $i \in V$. Furthermore, we delete the arc $(j, 1)$ (if present). In case that either j or 1 is isolated we know that the maximum flow has a value of $x_{j1} < 1$. The shores of the cut are trivially defined by the isolated node. If this is not successful, a maxflow algorithm is called on the shrunken digraph.

The separation of weak σ- (resp. (π,σ)-) inequalities follows a similar line. For all $j \in V$ (resp. $(i,j) \in R$) nodes in $\sigma(j)$ (resp. $\pi(i) \cup \sigma(j)$) are deleted and we try to send one unit of flow from 1 to j (resp. from i to j). The above reductions are applied as well.

Precedence Cycle Breaking Inequalities. The heuristic separation procedure (already outlined in [7]) is based on shrinking node sets that satisfy the subtour elimination inequality with equality. Simultaneously, the same node set is shrunk in the precedence digraph. It may happen that after shrinking infeasible arcs occur (e.g., cycles in the precedence graph). These structures correspond to violated inequalities. The procedure can be outlined as follows.

Input : $D = (V, A), P = (V, R)$ and a fractional LP-solution $\bar{x}$
Output : violated pcb-inequality or
violated inequality of type (14) and (15)
(if any such inequality is found)

1. Initialize: Set $m = n, S_i = \{i\}$ for all $i \in V, \tilde{y}_{ij} = \bar{x}_{ij}, \tilde{D} = D, \tilde{P} = P$.
2. Find a set $S \subseteq \tilde{V}$, s.t. $|S| \geq 2$ and $\tilde{y}(S) = |S| - 1$.
   If found : shrink S to a single node, update $\tilde{D} = (\tilde{V}, \tilde{A})$, $\tilde{P} = (\tilde{V}, \tilde{R})$, m and y,
   else STOP
3. If $\tilde{R}$ contains a cycle, then a violated precedence cycle breaking inequality (12) is found.
4. If $\tilde{y}_{ij} > 0$, for some $(i,j) \in \tilde{A}$ s.t. $(i,j) \in \tilde{R}_1$ (resp. $(i,j) \in \tilde{R}_2$), then a violated inequality of type (14) (resp. (15)) is found.
5. Goto 2.

One of the central steps in this procedure is how to detect the saturated subtour elimination inequality in step 2. In a first step, we shrink all arcs with value $\tilde{y}_{ij} = 1$. If no violated inequality is found, we shrink node sets of size 2, i.e., $\{i,j\} \in \tilde{V}$ with $\tilde{y}_{ij} + \tilde{y}_{ji} = 1$. These nodes are detected by enumeration. So far, larger node sets are not considered.

Strengthened T_k-inequalities. The strengthened T_k-inequalities are heuristically separated. We use the following straightforward heuristic separation procedure to detect such violated inequalities. The idea is that we try to "guess" nodes p, w, q and a node set W that are likely to violate a (strengthened) T_k-inequality.

In the first step, node sets S with $|S| \geq 2$ and $x(A(S)) = |S| - 1$ are determined. This is done by enumerating nodes $i, j \in V$ with $x_{ij} + x_{ji} = 1$, shrinking these nodes to a single node and iteratively applying this enumeration procedure to the resulting shrunken digraph. Each shrinking that is performed is stored in a shrinking digraph $D_s = (V, A_s)$, where an arc $(i,j) \in A_s$ occurs whenever clusters i and j are shrunk to a new cluster j. The iteration in which the arc is generated is stored as the arc label of (i,j). The outdegree of each node is at most 1, whereas more than one arc can enter a node. For a given node $i \in V$ node sets S_k containing i and satisfying $x(A(S_k)) = |S_k| - 1$ can easily be detected.

In the second step, promising node triples for the T_k inequality are enumerated. In order to keep computation time moderate we only consider nodes p, w, q, such that $x_{pw} + x_{pq} + x_{wq}$ is between some limiting values. Computational tests show that it is most likely to find (within

reasonable time) violated T_k-inequalities if $1.33 \leq x_{pw} + x_{pq} + x_{wq} \leq 1.75$. Furthermore, we generate just n such triples.

Finally, it is checked if a combination of the triples (p, w, q) and S_k violates a strengthened T_k-inequality.

2-matching inequalities. So far no separation procedure for strengthened 2-matching inequalities is known that incorporates the precedence structure directly. Therefore, we apply the heuristic separation procedure by Padberg and Rinaldi [22], that is based on the exact separation algorithm by Padberg and Rao [23]. If a violated 2-matching inequality is found, all strengthened versions (21)–(27) are checked. The most violated inequality is added.

SD- and D_k-inequalities. So far, we only add the basic ATSP version of the inequalities. We use the FORTRAN implementation of the separation procedure that was supplied by Fischetti and Toth and is described in [13].

Separation strategy. The separation procedures described above are called in a hierarchical order:

Input : Incumbent LP-solution $\bar{x}$.
Output : An inequality $ax \leq a_0$ violated by $\bar{x}$, i.e., $a\bar{x} > a_0$, if it can be found.

1. Search for violated pool inequalities
2. Exact separation of subtour elimination inequalities
3. Exact separation of D_k-inequalities
4. Heuristic separation of SD-inequalities
5. Heuristic separation of π-inequalities
6. Heuristic shrinking separation procedure for precedence cycle breaking inequalities
7. Heuristic separation of (π, σ)-inequalities
8. Heuristic separation of σ-inequalities
9. Heuristic separation of strengthened 2-matching inequalities
10. Heuristic separation of strengthened T_k-inequalities

If one of the steps is successful, i.e., a violated inequality is found, no other separation procedure is called.

4.3 Variable fixing

In several parts of the algorithm we aim at fixing variables either to 0 or 1. We use *reduced cost criteria* as well as criteria based on *logical implications*.

Reduced cost criteria allow to fix nonbasic active variables at their current value. To be more precise, suppose we are given a global lower bound *glb*, a global upper bound *gub*, and a nonbasic variable x_{ij} with associated reduced cost r_{ij}. In case that $x_{ij} = 0$ and $\lceil glb + r_{ij} \rceil \geq gub$ we are allowed to fix variable x_{ij} to zero, whereas $x_{ij} = 1$ and $\lceil glb - r_{ij} \rceil \geq gub$ imply that x_{ij} can be fixed to one. (Recall that we assumed to work with nonnegative, integer costs c_{ij}.)

Logical implications

Computational tests showed that variable fixing (resp. setting) due to logical implications is an important part of an efficient implementation, because it allows to reduce the problem dimension dramatically.

When a variable x_{ij} is fixed (set) to 1, it is possible to fix (set) a number of other variables. These variables include those corresponding to the reverse arc (j, i), to all other arcs with tail in i, resp. head in j, but as well variables x_{kl} for which we know that not both arcs (i, j) and (k, l) can be present in a feasible solution without violating a precedence relationship. We obtain the following implications:

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

After fixing all variables to zero it may happen that there exists nodes $i \in V$ such that $|\delta^-(i)| = 1$ (resp. $|\delta^+(i)| = 1$). Because at least one arc entering (resp. leaving) i has to be used, the corresponding variable can be fixed to 1.

Furthermore, if x_{ij} is fixed to 1, we update the precedence digraph $P = (V, R)$. On the one hand, we know that i precedes j and we can add the precedence relationship $i \prec j$ to the precedence digraph (if not yet present). On the other hand, node i (resp. j) "inherits" all the predecessors and successors of node j (resp. i). Afterwards, the transitive closure is updated and variables may be fixed (set) to zero due to Lemma 1. If after this update a new fixed node occurs, we decompose this problem into two subproblems and handle them separately.

4.4 Further implementation details

Preprocessing. In a preprocessing step we eliminate all arcs that cannot be present in any feasible tour. These are

- arcs (i, k) with $(i, j) \in R$ and $(j, k) \in R$
- arcs $(1, j)$ with $(i, j) \in R$
- arcs $(i, 1)$ with $(i, j) \in R$

Furthermore, we check if *fixed nodes* are present, i.e., nodes $i \in V$ such that $|\pi(i)| + |\sigma(i)| = n - 2$ holds. If such nodes exist, it is possible to decompose the instance into two smaller instances by "splitting" it at node i and to handle them separately.

Initial set of variables. All arcs that are not fixed in the preprocessing phase are stored explicitly in the feasible arc set. Since only a small subset of the variables will be nonzero in an optimal solution, we are not working on the whole set of variables, but choose a promising subset of variables, which we hope to contain an optimal solution. We have chosen the variables of the k-nearest neighbor digraph, which we also denote as *sparse digraph* $D_s = (V, A_s)$. We have tested several possible values of k and have finally chosen $k = 4$ (see also

the paragraph on Pricing). We run an *initial heuristic* to obtain a feasible solution. The main purpose of this heuristic is to make the sparse digraph Hamiltonian. The arcs of that tour are added to the sparse digraph. Only the variables corresponding to the arcs of the sparse digraph are considered in the initial linear program. The value of the global upper bound is initialized with the value of this feasible tour.

Initial linear program. We set up the initial linear program with the variables in the sparse digraph D_s, the degree constraints, and the trivial bounds $x_{ij} \geq 0$.

Pricing. A pricing step has to be performed before branching, if an infeasibility is detected, and if a tailing-off phenomenon is observed. Recall that basically the variables corresponding to the k-nearest neighbor digraph are considered in the initial LP. The "sparse-digraph technique" keeps the size of the linear program "moderate" but makes it necessary to price out nonactive variables, that are not considered in the actual LP. The purpose is to check if the LP solution is valid for the complete digraph, i.e., if all nonactive variables price out correctly.

Since this may be very time consuming, the *pricing* is performed in a hierarchical way. First the variables in a so-called *reserve digraph* $D_r = (V, A_r)$ are considered, before performing a pricing on the complete set of variables. We have chosen D_r to be the m-nearest neighbor digraph for $m > k$. Arcs already contained in A_s are not considered in A_r. If the reserve-graph-pricing is successful, the complete pricing is omitted.

The computational experiments documented in [2] showed that in the current implementation the choice of m and k only has a minor influence on the overall computation time. We tested values for $k \in \{2, 3, 4, 5, 6, 7\}$ and $m \in \{5, 6, 7, 8, 9, 10\}$ leading to more or less the same overall computation times. We decided to choose $k = 4$ and $m = 7$ as default values.

Furthermore, the pricing step is performed every pricing_freq iterations. We have chosen pricing_freq = 10 as default value. This assures that variables needed in an optimal solution enter the LP early.

Tailing off. If during the last k iterations of the cutting plane phase no significant improvement (of at least $p\%$) is achieved, a pricing step is performed. If this is not successful, i.e., no variables are added, we leave the cutting plane phase and branch. We use the default values $k = 5$ and $p = 2.5 \cdot 10^{-5}$.

Branching. Branching is performed on variables only. In the branching step a fractional variable is chosen and two new subproblems are generated by setting the value of the variable to 0, resp. to 1. We have chosen the following branching strategy that is supported by the branch&cut framework: Choose the variable with value closest to 0.5; in case that several such variables exist, the one with the highest cost-coefficient is chosen.

Enumeration strategy. In order to keep the branch&cut tree small, the way in which the subproblems in the tree are processed is very important for an efficient implementation. In our implementation we tested three different strategies that are supported by the branch&cut framework: depth-first-search (DFS), breadth-first-search (BRFS), and best-first-search (BEFS). The computational results on some benchmark problems showed that DFS is not a good strategy, because the "risk" of spending too much time in a branch of the tree that

Key to Tables 1–4:

problem	:	Name of the problem instance.
n	:	Number of nodes.
$\|R\|$	:	Number of precedence relationships (without transitively derived precedences).
Solution	:	Value of the found solution. If the instance is not solved to optimality, the global lower bound *glb* and global upper bound *gub* is given in the form *[glb,gub]*.
qual.	:	Quality of final feasible solution calculated by $\frac{gub-glb}{glb} \cdot 100$ (if the problem instance is solved to optimality it is marked with "–").
Heur.	:	Value of the solution obtained by the initial heuristic.
BC-root	:	Quality of the solution at the root LP.
...bounds	:	Lower and upper bound at the root LP. If the problem instance is solved to optimality at the root, it is marked with "–".
...GAP(lb/ub)	:	Optimality gap at the root node calculated separately for the lower and upper bound. Let lb_{root} (resp. ub_{root}) denote the lower bound (resp. upper bound) before branching. Then the optimality gap at the root is defined as $\frac{gub-lb_{root}}{lb_{root}} \cdot 100$ (resp. $\frac{ub_{root}-glb}{glb} \cdot 100$).
BC-tree ...	:	Information on the branch&cut tree.
... # N	:	Number of generated nodes in the branch&cut tree (except the root node).
... level	:	Depth of the branch&cut tree.
#cuts	:	Number of generated cutting planes.
#LPs	:	Number of linear programs that had to be solved.
CPU	:	CPU time needed to solve the problem in format *min:sec*. If the problem instance cannot be solved to optimality within a certain time limit, this is marked by giving the CPU-time after which the run is stopped (all computation times for SUN SPARC 10).

is useless for the computation of the bounds is too high. BRFS slightly outperforms BEFS (see Ascheuer [2]) as it tends to give more "stable" results. Thus, we decided to use BRFS as a default strategy.

5 Computational Results

The branch&cut-algorithm described in the previous section is coded in `C` on a SUN Ultra-SPARC Model 140 with 128 MB main memory under SUN OS 5.6. We use a preliminary version of the general purpose branch&cut-framework ABACUS that is explained in [18], the LP-solver CPLEX 5.0, and the FORTRAN implementations of the separation routines for SD- and D_k-inequalities by Fischetti and Toth [13]. The code is tested on several classes of problem instances that contain real–life data, randomly generated data, and a combination of both, involving the benchmark problems for the SOP contained in TSPLIB [24].

For all instances we allow a maximum computation time of $5 \cdot \lceil \frac{n}{100} \rceil$ hours of CPU-time and they are all run with the default parameter settings that we have described in the previous section. It is obvious that better results on some of the instances may be achieved by fine-tuning the parameters for these instances. But the aim of the computational experiments is to evaluate parameter settings that give satisfactory results for a wide range of problem instances. If an instance cannot be solved to optimality within the given time limit, we give

the lower and upper bounds found by the algorithm. First, we briefly describe the classes of problem instances we use, afterwards we discuss the results achieved on these instances. The results are summarized in Tables 1–4.

Table 1: Results on real–life instances

						BC–root			BC–tree			
Problem	n	$\|R\|$	Solution	qual.	Heur.	bounds	GAP		#N, Level	# cuts	#LPs	CPU
ESC07	9	6	2125	—	2550	–	–	–	0, 0	4	3	0:00
ESC11	13	3	2075	—	2129	–	–	–	0, 0	13	7	0:00
ESC12	14	7	1675	—	1760	–	–	–	0, 0	34	34	0:00
ESC14	16	12	2125	—	2125	–	–	–	0, 0	20	10	0:00
ESC25	27	9	1681	—	2372	[1642, 1684]	2.38	2.56	4, 2	85	93	0:01
ESC47	49	10	1288	—	2609	[1247, 1370]	3.29	9.86	40, 6	580	643	0:28
ESC63	65	95	62	—	63	–	–	–	0, 0	4	3	0:00
ESC78	80	77	18230	—	20130	–	–	–	0, 0	41	16	0:01
ESC98	100	84	2125	—	2125	–	–	–	0, 0	55	17	0:05
rbg019a	21	43	198	—	198	[197, 198]	0.51	0.51	4, 2	5	9	0:00
rbg019b	21	57	199	—	214	–	–	–	0, 0	4	2	0:00
rbg021a	21	68	158	—	183	–	–	–	0, 0	20	13	0:00
rbg023a	23	79	155	—	193	–	–	–	0, 0	42	22	0:00
rbg029a	29	76	217	—	248	[216, 218]	0.46	0.93	8, 3	88	56	0:01
rbg048a	50	192	351	—	396	[347, 351]	1.15	1.15	10, 5	493	235	0:21
rbg049a	51	241	355	—	425	–	–	–	0, 0	37	13	0:00
rbg050a	52	225	400	—	460	–	–	–	0, 0	157	66	0:03
rbg050b	52	258	397	—	465	–	–	–	0, 0	55	16	0:00
rbg050c	52	256	467	—	497	[465, 467]	0.43	0.43	12, 4	358	91	0:03
rbg068a	68	249	609	—	689	–	–	–	0, 0	57	19	0:01
rbg088a	90	547	1130	—	1245	[1124, 1133]	0.53	0.80	26, 4	1734	289	0:40
rbg092a	94	573	[1035, 1037]	0.19	1098	[1032, 1037]	0.48	0.48	11684,19	15609	29981	–*
rbg094a	96	457	1336	—	1401	–	–	–	0, 0	24	9	0:00
rbg105a	107	682	[996, 1023]	2.71	1224	[990, 1030]	3.33	4.04	24718,14	11594	54110	–*
rbg109a	111	622	1038	—	1117	[1027, 1042]	1.07	1.46	16734,34	4952	34703	232:59
rbg113a	113	305	1432	—	1537	[1431, 1432]	0.07	0.07	2, 1	95	36	0:05
rbg117a	117	312	1494	—	1609	–	–	–	0, 0	74	22	0:04
rbg118a	118	312	1423	—	1551	–	–	–	0, 0	390	103	0:27
rbg124a	124	351	1361	—	1520	–	–	–	0, 0	70	10	0:02
rbg126a	126	369	1381	—	1587	[1376, 1384]	0.36	0.58	1098,19	2273	3734	17:55
rbg143a	143	359	1765	—	1966	–	–	–	0, 0	46	14	0:03
rbg148a	148	976	[1396, 1399]	0.21	1554	[1376, 1407]	1.67	2.25	10054,18	16003	26503	–*
rbg150a	152	952	[1748, 1750]	0.11	1858	[1745, 1750]	0.29	0.29	12230,13	19263	31151	–*
rbg161a	161	472	1962	—	2178	–	–	–	0, 0	138	27	0:11
rbg174a	176	1113	[2029, 2033]	0.20	2193	[2022, 2037]	0.54	0.74	8376,12	16741	21490	–*
rbg190a	190	725	[2229, 2241]	0.54	2442	[2209, 2251]	1.45	1.90	8372,13	20744	29303	–*
rbg219a	219	799	[2530, 2544]	0.55	2767	[2512, 2551]	1.27	1.55	9574,13	24366	32566	–*
rbg247a	247	702	3062	—	3379	–	–	–	0, 0	284	55	1:16
rbg253a	255	1721	[2929, 2951]	0.75	3163	[2918, 2961]	1.13	1.47	5074,11	16801	16255	–*
rbg285a	285	820	3482	—	3857	[3479, 3483]	0.09	0.11	442,15	1770	1584	42:51
rbg323a	325	2412	[3137, 3141]	0.13	3381	[3126, 3157]	0.48	0.99	2146,10	24360	11607	–*
rbg341a	343	2542	[2528, 2582]	2.14	3136	[2504, 2651]	3.12	5.87	2536,10	13900	9653	–*
rbg358a	360	3239	[2527, 2568]	1.62	3185	[2499, 2601]	2.76	4.08	1094, 9	14480	6536	–*
rbg378a	380	3069	[2771, 2856]	3.07	3431	[2731, 2881]	4.58	5.49	1120, 9	14616	6311	–*

–*: time limit exceeded

Real-life data. The problem instances `ESC07`–`ESC98` are production data from IBM, that were supplied by Escudero. They correspond to the instances `P1`–`P9` as given in Ascheuer et al. [4]. The instances `rbg019a`–`rbg378a` correspond to data that was obtained from the routing of a stacker crane in an automatic storage system (for more details see Ascheuer [2]).

Randomly generated problem instances. All instances `prob.*` are randomly generated. `prob.7` (in TSPLIB `prob.100`) turned out to be one of the hardest instances we considered, although the precedence relationships are easily structured. The instance contains 41 disjoint pair of nodes u_i, v_i such that $u_i \prec v_i$ holds. Nevertheless, it seems to be nontrivial to find good feasible solutions. It is not known if the best feasible solution (1385) is close to the value of the optimal solution, or if the best lower bound (approx. 1035) is the tighter bound. In order to study the reasons for this unsatisfactory behavior we constructed several other instances by just considering the first k nodes of this instance (`prob.7.k`).

Table 2: Solution of random problem instances

						BC–root			BC–tree			
Problem	n	$\|R\|$	Solution	qual.	Heur.	bounds	GAP		#N, Level	# cuts	#LPs	CPU
prob.1	11	6	38	—	38	–	–	–	0, 0	6	3	0:00
prob.2	11	6	287	—	287	–	–	–	0, 0	2	2	0:00
prob.3	22	4	218	—	272	[217, 218]	0.46	0.46	2, 1	85	61	0:00
prob.4	42	0	225	—	233	–	–	–	0, 0	23	3	0:00
prob.5	42	10	243	—	285	[227, 254]	7.05	11.89	348,14	3715	5443	3:55
prob.6	82	5	614	—	859	[614, 643]	0.00	4.72	4, 1	259	57	0:10
prob.7	100	41	[1025, 1448]	41.27	1990	[1004, 1669]	44.22	66.24	6756,11	48778	42546	–*
prob.7.30	30	5	898	—	1260	[874, 927]	2.75	6.06	14, 4	216	203	0:03
prob.7.35	35	7	957	—	957	[910, 957]	5.16	5.16	94, 9	1124	579	0:18
prob.7.40	40	10	1071	—	1587	[1054, 1146]	1.61	8.73	16, 5	379	223	0:07
prob.7.45	45	11	974	—	974	[932, 974]	4.51	4.51	120,10	1643	830	0:42
prob.7.50	50	14	879	—	1651	[866, 920]	1.50	6.24	28, 4	644	379	0:18
prob.7.55	55	16	882	—	1471	[847, 967]	4.13	14.17	1740,12	15908	10938	22:03
prob.7.60	60	20	[907, 1024]	12.90	1561	[855, 1242]	19.77	45.26	13416,12	83788	93316	–*
prob.7.65	65	23	[899, 1049]	16.69	1382	[884, 1291]	18.67	46.04	13080,12	71182	81284	–*
prob.7.70	70	27	[871, 1005]	15.38	1625	[853, 1245]	17.82	45.96	10696,12	75501	75066	–*

–*: time limit exceeded

Pure ATSP instances. Since the ATSP is a special case of the SOP where the set of precedence relationships is empty, we obtain as a side product an algorithm to solve asymmetric TSPs to optimality. We run the algorithm with the (hard) ATSP instances contained in TSPLIB [24] (varying from 17 to 100 nodes). Furthermore, we run several large-scale instances that have been derived from the stacker crane application (problem instances `rbg*.0`).

ATSP instances with random precedences. It turned out that the ATSPs that are the basis for the real–life problems `rbg*` are easily solvable. In order to check what influence additional precedence constraints have on hard ATSP instances we perform experiments on the ATSP instances contained in TSPLIB [24]. For that purpose we randomly generate precedence digraphs $P = (V, R)$ and add them to the ATSP instances.

The random precedence digraph $P = (V, R)$ is constructed in the following way: Initially we set $R = \emptyset$. For each problem instance two lists l_1, l_2 of k random integers in the interval $[1, n]$ are generated. List l_1 corresponds to the tails, list l_2 to the heads of potential arcs $(i, j) \in R$. Starting from the first entry of the list we add an arc $(l_1[i], l_2[i])$ to R if $l_1[i] \neq l_2[i]$, $(l_1[i], l_2[i]) \notin R$, and $R \cup (l_1[i], l_2[i])$ remains acyclic — otherwise this arc is skipped. We generate problem instances for $k = \frac{n}{4}, \frac{n}{2}, n, 2 \cdot n$. Because some infeasible arcs are generated, the actual number of added precedences is smaller than k.

Table 3: Solution of asymmetric TSP instances

Problem	n	Opt.	Heur.	BC–root bounds	BC–root GAP		BC–tree #N, Level	# cuts	#LPs	CPU
br17	18	39	39	–	–	–	0, 0	9	7	0:00
p43	44	28100	28130	[28058,28100]	0.15	0.15	170,23	1115	808	0:52
ry48p	49	14422	14732	[14351,14519]	0.49	1.17	8, 3	176	255	0:15
ft53	54	6905	7667	–	–	–	0, 0	25	21	0:06
ft70	71	38673	39653	[38660,38673]	0.03	0.03	6, 3	30	38	0:23
kro124p	101	36230	39468	[36217,36230]	0.04	0.04	2, 1	163	86	2:25
rbg323.0	324	1326	1327	–	–	–	0, 0	8	10	9:54
rbg341.0	342	1116	1120	–	–	–	0, 0	1	8	10:39
rbg358.0	359	1163	1165	–	–	–	0, 0	5	13	12:10
rbg378.0	379	1633	1634	–	–	–	0, 0	12	14	16:48
rbg399.0	400	2048	2048	–	–	–	0, 0	0	5	16:22
rbg403.0	404	2465	2465	–	–	–	0, 0	0	5	13:18
rbg416.0	417	2126	2136	–	–	–	0, 0	1	7	29:02
rbg423.0	424	2065	2066	–	–	–	0, 0	0	6	20:38
rbg443.0	444	2720	2720	–	–	–	0, 0	0	6	19:37

If `name` is the name of the TSPLIB-instance then

`name.1` is used for additional $k = \frac{n}{4}$ potential precedence relationships,
`name.2` is used for additional $k = \frac{n}{2}$ potential precedence relationships,
`name.3` is used for additional $k = n$ potential precedence relationships,
`name.4` is used for additional $k = 2 \cdot n$ potential precedence relationships.

On the one hand, most of the small and medium sized problem instances can be solved to optimality within a reasonable amount of computation time (Table 1). On the other hand, there exist several other instances for which the optimality gap is pretty large even after several hours of computation time (see column Solution). These instances are mainly among the TSPLIB-instances with random precedences (see Table 4).

For all of the real–life instances in Table 1 good lower and upper bounds on the value of an optimal solution can be found even in the root of the branch&cut tree. For all instances, except `ESC47`, the gap of the bounds is in a 4%-range. Both lower and upper bound differ by the same order of magnitude from the value of an optimal solution (see Column *BC-root...GAP*).

A similar behavior can be observed for the instances summarized in Table 4. Nevertheless, the final GAP for these instances is larger. Moreover, the gap before branching is larger than for the other classes of instances. As a consequence, the major part of these instances cannot be solved to optimality within the given time limit (although they are smaller in size than many of the real–life instances). Although several separation routines are available and a large number of cuts are generated, this is not sufficient. This is perhaps due to the fact that the generated inequalities are only valid inequalities but we do not know if they are facet defining, i.e., they are not strong enough from a computational point of view. As for the TSP the size of the instance, typically indicated by the number of nodes, is not the only indicator for the "hardness" of an instance.

It seems worth mentioning that only for very few instances the initial heuristics yield good feasible solutions. For most of the considered instances better solutions are found "on the fly" by the LP-exploitation heuristic. The results of the initial heuristic are getting worse the more precedences are involved (e.g., instances `rbg*`). This indicates that the starting heuristics do not handle the precedence structure properly.

All ATSP instances can be solved to optimality in a short computation time (Table 3).

Table 4: Solution of TSPLIB instances with random precedences

Problem	n	$\|R\|$	Solution	qual.	Heur.	BC–root bounds	BC–root GAP		BC–tree #N, Level	# cuts	#LPs	CPU
br17.1	18	4	41	—	41	[39, 41]	5.13	5.13	10, 5	100	84	0:01
br17.2	18	8	47	—	49	[40, 47]	17.50	17.50	168,25	1099	1065	0:16
br17.3	18	14	49	—	49	[43, 49]	13.95	13.95	6, 3	104	72	0:00
br17.4	18	17	76	—	76	–	–	–	0, 0	31	14	0:00
p43.1	44	9	28140	—	28280	[28063,28170]	0.27	0.38	1132,27	3695	3788	6:21
p43.2	44	20	[28319,28480]	0.57	28630	[28109,28500]	1.32	1.39	7670,14	32859	40238	–*
p43.3	44	37	[28553,28835]	0.99	29255	[28160,28890]	2.40	2.59	5876,12	25618	35939	–*
p43.4	44	50	[65497,83005]	26.73	83525	[55884,83050]	48.53	48.61	12604,25	15949	50570	–*
ry48p.1	49	11	15805	—	16984	[14969,15994]	5.58	6.85	7456,29	19575	72208	208:03
ry48p.2	49	23	[15587,16666]	6.92	17562	[15117,16940]	10.25	12.06	7754,12	25642	78473	–*
ry48p.3	49	42	[17813,19894]	11.68	22406	[17248,20370]	15.34	18.10	5836,11	14741	59392	–*
ry48p.4	49	58	[29616,31446]	6.18	34467	[27165,31446]	15.76	15.76	11322,13	9109	59114	–*
ft53.1	54	12	7531	—	8723	[7186, 7695]	4.80	7.08	10084,32	7418	45925	112:48
ft53.2	54	25	[7594, 8054]	6.06	10674	[7419, 8691]	8.56	17.15	14950,13	10342	70207	–*
ft53.3	54	48	[9253,10262]	10.90	13743	[9115,10748]	12.58	17.92	4190,11	12290	47152	–*
ft53.4	54	63	[14124,14425]	2.13	16231	[13871,14425]	3.99	3.99	13044,17	10239	60653	–*
ft70.1	71	17	39313	—	41960	[39168,39617]	0.37	1.15	348,10	1150	2327	6:03
ft70.2	71	35	[39843,40485]	1.61	43883	[39417,41628]	2.71	5.61	17080,13	10244	65637	–*
ft70.3	71	68	[41413,42884]	3.55	48659	[40646,43563]	5.51	7.18	7826,11	12178	51554	–*
ft70.4	71	86	[52298,53583]	2.46	60806	[50611,53627]	5.87	5.96	5892,11	12010	37201	–*
kro124p.1	101	25	[37861,39982]	5.60	46063	[36538,42814]	9.43	17.18	5310,11	23082	54957	–*
kro124p.2	101	49	[38809,42055]	8.36	48330	[37591,44064]	11.88	17.22	6112,11	18643	53928	–*
kro124p.3	101	97	[41451,51123]	23.33	56198	[39323,53389]	30.01	35.77	5416,11	11125	47211	–*
kro124p.4	101	131	[65445,76103]	16.29	91579	[56516,79377]	34.66	40.45	3582,10	10394	28945	–*

–*: time limit exceeded

These results indicate that the precedence constrained instances are "more difficult" than pure ATSP instances of the same size. The `rbg`-instances are easily solvable. All of them can be solved just after a few iterations and very often only subtour elimination inequalities suffice to solve the instances. The achieved results are comparable to the best computation times documented by Fischetti and Toth [13]. This is not surprising as more or less the same separation routines are used.

Table 5 shows results on a subset of all test instances indicating the strength of the various separation routines. To concentrate on the influence of the separation procedures on the lower bound only, an optimal solution was supplied as input sequence. Column ALL gives the results when all separation routines are used. The first entry corresponds to the optimality gap at the root node ($\frac{opt-glb}{glb} \cdot 100$), the second entry to the overall time (*min:sec*) to solve the instance to optimality. In the last row gaps and computing times are summed up.

It turns out that the most efficient separation routines are those for π-, σ-, and (π, σ)-inequalities. Computing times considerably increase if these procedures are not used. They

Table 5: Results for separation routines

	All		without π		without σ		without (π,σ)		without 2M		without D_k		without SD		without T_k	
ESC25	0.18	0:00	0.60	0:00	2.19	0:01	0.36	0:00	0.18	0:00	0.06	0:00	0.30	0:00	0.18	0:00
ESC47	3.21	0:15	3.37	0:15	3.29	0:16	3.45	0:26	3.21	0:16	3.29	0:17	3.29	0:13	3.21	0:15
ft70.1	0.37	0:35	0.44	0:45	0.41	2:32	0.37	0:34	0.37	0:38	0.38	0:29	0.38	0:45	0.37	0:47
prob.7.35	5.16	0:18	5.28	0:31	5.28	0:37	5.16	0:18	5.16	0:20	5.05	0:17	5.16	0:14	5.16	0:18
prob.7.45	4.28	0:42	4.28	1:11	6.22	1:41	4.28	0:41	6.10	1:14	5.98	0:48	4.28	0:20	4.28	0:44
rbg048a	1.45	0:11	0.57	12:14	1.15	0:29	1.15	0:51	1.45	0:20	1.45	0:15	1.45	0:14	1.15	2:36
rbg118a	0.00	0:27	0.00	0:11	0.07	0:23	0.07	2:02	0.00	0:27	0.00	0:32	0.07	0:26	0.00	0:28
rbg247a	0.00	1:16	0.00	0:58	0.00	0:54	0.03	1:19	0.00	1:11	0.00	1:18	0.00	1:15	0.00	1:26
$\sum$	14.65	3:44	14.54	16:05	18.61	6:53	14.87	6:11	16.47	4:26	16.21	3:56	14.93	3:27	14.35	6:34

generate many cuts that considerably improve the objective function value (see, e.g., the gap for the σ–separation).. Furthermore, we emphasize the computational power of strengthening generated subtour elimination inequalities. Especially for problem instances with dense precedence structure, very seldomly pure subtour elimination inequalities are added as cutting planes. Also, very few cuts can be strengthened to simple pcb-inequalities. Moreover, the separation routines for D_k- and SD-inequalities as well as the shrinking procedure for identifying violated precedence cycle breaking inequalities prove to be very helpful. The separation routines for 2-matching inequalities and strengthened T_k-inequalities only play a minor role in solving instances to optimality.

6 Conclusions

The *sequential ordering problem* (SOP), or equivalently *asymmetric traveling salesman problem with precedence constraints* is an important basic model for many applications in scheduling and routing.

In this paper we have described the implementation of a branch&cut-algorithm that solves problem instances to optimality or gives quality bounds on the value of the best solution found. This is in contrast to pure heuristic methods.

Computational experiments with both real–life data obtained from industrial applications as well as randomly generated instances show that the branch&cut-algorithm is capable of solving medium sized instances of more than 200 nodes within a few minutes of CPU-time to provable optimality. Our method clearly outperforms other exact methods published so far.

An analysis of the bounds obtained before branching shows that for most problem instances very good lower bounds are obtained. Sometimes it is difficult to find a feasible solution with that value. The results on some instances, especially large scale instances with dense precedence structure, show that the heuristics sometimes fail to construct good solutions. From our point of view, this is one of the main reasons that most of the larger instances cannot be solved to optimality. Recently, alternative heuristic algorithms have been proposed [8, 14] that seem to give good results for these type of problem instances. Their use within the branch&cut-algorithm should be investigated.

Furthermore, we believe that the derivation of new classes of inequalities and the incorporation of further separation routines will improve the computation times and will lead to optimal solutions for some of the unsolved instances. This is left to further research.

Acknowledgements

Our research was partially supported by the Science Program SC1-CT91-620 of the EEC. We are grateful to Stefan Thienel who provided the implementation of the branch&cut framework. We also thank Matteo Fischetti and Paolo Toth for having provided us with their FORTRAN code for the separation of the D_k- and SD-inequalities.

References

[1] N. Ascheuer. Ein Schnittebenenverfahren für ein Reihenfolgeproblem in der flexiblen Fertigung. Master's thesis, Universität Augsburg, Germany, 1989.

[2] N. Ascheuer. *Hamiltonian Path Problems in the On-line Optimization of Flexible Manufacturing Systems*. PhD thesis Tech. Univ. Berlin, 1995. Avail. at URL `http://www.zib.de/ZIBbib/Publications/`,

[3] N. Ascheuer, L. Escudero, M. Grötschel, and M. Stoer. On identifying in polynomial time violated subtour elimination and precedence forcing constraints for the sequential ordering problem. In Kannan, R. and Pulleyblank,W.R., editors, *Integer Programming and Combinatorial Optimization*, pages 19–28. University of Waterloo, Waterloo, Ontario, 1990.

[4] N. Ascheuer, L. Escudero, M. Grötschel, and M. Stoer. A cutting plane approach to the sequential ordering problem (with applications to job scheduling in manufacturing). *SIAM Journal on Optimization*, 3:25–42, 1993.

[5] N. Ascheuer, M. Jünger, and G. Reinelt. Heuristic algorithms for the ATSP with precedence constraints – a computational comparison. Technical report, ZIB Berlin, 1998. (To appear).

[6] 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.

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

[8] S. Chen and S. Smith. Commonality and genetic algorithms. Technical Report Technical Report CMU-RI-TR-96-27, Carnegie Mellon University, Pittsburgh, 1996.

[9] L.F. Escudero. An inexact algorithm for the sequential ordering problem. *European Journal of Operational Research*, 37:236–253, 1988.

[10] L.F. Escudero. On the implementation of an algorithm for improving a solution to the sequential ordering problem. *Trabajos de Investigacion-Operativa*, 3:117–140, 1988.

[11] L.F. Escudero, M. Guignard, and K. Malik. A Lagrangean relax–and–cut approach for the sequential ordering problem with precedence constraints. *Annals of Operations Research*, 50:219–237, 1994.

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

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

[14] L.M. Gambardella and M. Dorigo. HAS-SOP: Hybrid ant system for the sequential ordering problem. Technical Report Technical Report IDSIA-11-97, IDSIA, Lugano, Switzerland, 1997.

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

[16] 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.

[17] 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.

[18] M. Jünger, G. Reinelt, and S. Thienel. Provably good solutions for the traveling salesman problem. *Zeitschrift für Operations Research*, 22:83–95, 1998.

[19] 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`.

[20] M. Jünger, and S. Thienel. Introduction to ABACUS - A Branch And CUt System. *Operations Research Letters*, 40(2):183–217, 1994.

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

[22] M. Padberg and G. Rinaldi. Facet identification for the symmetric traveling salesman polytope. *Mathematical Programming*, 47:219–257, 1990.

[23] M. Padberg and M.R. Rao. Odd minimum cut-sets and b-matchings. *Math. of OR*, 7:67–80, 1982.

[24] G. Reinelt. TSPLIB – a traveling salesman problem library. *ORSA Journal on Computing*, 3:376–384, 1991. See
`http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/TSPLIB95/TSPLIB.html`.

[25] M. Timlin. Precedence constrained routing. Master's thesis, Department of Combinatorics and Optimization, University of Waterloo, 1989.

[26] M.T. Fiala Timlin and W.R. Pulleyblank. Precedence constrained routing and helicopter scheduling: Heuristic design. *Interfaces*, 22(3):100–111, May–June 1992.