A Branch-and-Cut Algorithm for the Undirected Selective Traveling Salesman Problem

Michel Gendreau, Gilbert Laporte, Frédéric Semet

Centre de recherche sur les transports, Université de Montréal, Case postale 6128, succursale “Centre-ville,” Montréal, Canada H3C 3J7

Received December 1995; accepted 26 May 1998

Abstract: The Selective Traveling Salesman Problem (STSP) is defined on a graph in which profits are associated with vertices and costs are associated with edges. Some vertices are compulsory. The aim is to construct a tour of maximal profit including all compulsory vertices and whose cost does not exceed a preset constant. We developed several classes of valid inequalities for the symmetric STSP and used them in a branch-and-cut algorithm. Depending on problem parameters, the proposed algorithm can solve instances involving up to 300 vertices.

Keywords: selective traveling salesman problem; orienteering; branch-and-cut algorithm

1. INTRODUCTION

The purpose of this article was to describe a branch-and-cut algorithm for the undirected Selective Traveling Salesman Problem (STSP). This problem is defined on a complete graph \( G = (V, E) \), where \( V = \{v_1, \ldots, v_n\} \) is a vertex set, \( v_1 \) is a depot, and \( E \) is an edge set. A non-negative profit \( p_i \) is associated with every vertex \( v_i \) and a nonnegative cost or distance matrix \( C = (c_{ij}) \) satisfying the triangle inequality is defined on \( E \). The STSP consists of determining a maximal profit Hamiltonian cycle on a subgraph of \( G \) containing a subset \( T \subseteq V \) of compulsory vertices (with \( v_1 \in T \)) and having a cost not exceeding a preset limit \( c \). Since it will never be profitable to visit vertices of \( V \setminus T \) with zero profit, we can assume that profits are positive for all \( v_i \in V \setminus T \). Finally, we assume that all profits are integer. Since all vertices of \( T \) must be included in the solution, their profits can be set equal to zero. If \( T = V \), then the STSP reduces to the Traveling Salesman Problem (TSP).

An application of the STSP arises in the delivery of home heating oil (Golden et al. [10]). In this problem, \( V \) is a set of customers requiring heating oil deliveries. The amount of fuel of each customer is supposed to stay above a certain level or resupply point in the tank. Using forecasts, a “measure of urgency” \( p_i \) can be determined for each customer. The problem associated with a vehicle is to maximize the sum of all \( p_i \) coefficients associated with customers visited during one day, subject to an upper limit on the length or duration of the delivery route. Another application is orienteering, a treasure-hunt game in which competitors collect rewards at various locations within a given time limit (Hayes and Norman [14]. Tsiligirides [23]). Hence, the STSP is also referred to as the Orienteering Problem, but some authors (e.g., [17]) use the expression STSP, which is more general.

A problem closely related to the STSP is the Prize
Collecting Traveling Salesman Problem (Fischetti and Toth [7], Balas [1]) in which the aim is to determine a least-cost cycle over a subset of vertices, subject to a lower limit on the accumulated profit. Göthe-Lundgren et al. [13] studied a variant of the STSP in which, in addition to \( p_i \), each vertex \( v_i \) has an associated weight \( q_i \).

Here, the aim was to determine a cycle minimizing the sum of routing costs minus the sum of profits, subject to an upper limit on the total weight collected. This model is used for the computation of nucleoli in cooperative games (Göthe-Lundgren et al. [12]). A different selective tour problem with a vertex covering specification was studied by Gendreau et al. [9]. Here, we are given a set \( T \) of vertices to be covered, in addition to \( V \). The aim was to construct a minimal-length tour through a subset of \( V \) such that every vertex of \( W \) is within a specified distance of a vertex on the tour.

The STSP is NP-hard, as can readily be shown by a reduction from the Hamiltonian Cycle Problem (Laporte and Martello [17]). Heuristics for the STSP with \( T = \{ v_i \} \) were proposed by Hayes and Norman [14]. Tsiligirdes [23], Golden et al. [10, 11], Keller [15], Laporte and Martello [17], Ramesh and Brown [21], Chao [2], as well as Chao et al. [3]. Laporte and Martello [17] developed an enumerative algorithm, while Ramesh et al. [22] proposed an exact Lagrangean relaxation-based method. Leifer and Rosenwein [19] added some valid inequalities for the formulation presented in Laporte and Martello [17].

In this article, we develop several classes of valid inequalities. These are then used within a branch-and-cut algorithm. We also describe two new heuristics used to provide an initial feasible solution. Furthermore, we generalize previous models by not restricting \( T \) to \( \{ v_i \} \).

The remainder of the paper is organized as follows: The model and the valid inequalities are presented in Section 2, followed by two heuristics in Section 3, by the branch-and-cut algorithm in Section 4, and by computational results in Section 5. We conclude in Section 6.

2. THE MODEL

To formulate the STSP as an integer linear program, define binary variables \( y_i \), equal to 1 if and only if vertex \( v_i \) belongs to the solution, and binary variables \( x_{ij} (i < j) \), equal to 1 if and only if edge \( (v_i, v_j) \) appears on the optimal tour. In what follows, assume that the solution contains at least three vertices since simple tours of the form \( (v_1, v_2, v_3) \) can readily be enumerated for \( k = 2, \ldots, n \). We use the following model for the STSP. It extends the formulation proposed in [17] to account for compulsory vertices:

\begin{equation}
\text{Maximize } z = \sum_{v_i \in V \setminus T} p_i y_i, \tag{1}
\end{equation}

subject to

\begin{align*}
\sum_{i < j} c_{ij} x_{ij} & \leq c \tag{2} \\
\sum_{i < k} x_{ik} + \sum_{j > k} x_{kj} &= 2y_k \quad (v_k \in V) \tag{3} \\
\sum_{v_i \in S} x_{ij} & \geq 2y_i \quad \forall i \in S \setminus V \setminus S \tag{4}
\end{align*}

\((S \subseteq V, 2 \leq |S| \leq n - 2, T \setminus S \neq \emptyset, v_i \in S)\)

In this model, the upper limit on the cost of the cycle is imposed through constraint (2). Constraints (3) are degree constraints. Constraints (4) are connectivity constraints that force the presence of at least two edges between \( S \) and its complement whenever vertex \( v_i \in S \) belongs to the solution and there exist some compulsory vertices outside \( S \). Constraints (5), (6), and (7) are integrality constraints. Note that this formulation is valid as long as \( C \) satisfies the triangle inequality for otherwise some vertices could have a degree larger than 2.

This model can be reinforced through the introduction of valid inequalities. Propositions 1, 2, and 3 are trivially established.

**Proposition 1.** If \( v_i \) or \( v_j \in V \setminus T \), then the constraints

\begin{equation}
x_{ij} \leq y_i \quad \text{and} \quad x_{ji} \leq y_j \tag{8}
\end{equation}

and

\begin{equation}
y_i + y_j \leq 1 \quad (c_{ii} + c_{ij} + c_{jj} > c) \tag{9}
\end{equation}

are valid inequalities for (STSP).

Constraints (8) and (9) were proposed by Leifer and Rosenwein [19] and by Keller [15], respectively. Constraint (9) can easily be generalized as follows:

**Proposition 2.** Let \( S \subseteq V \setminus T \) and let \( \sigma \) denote a lower bound on the length of an optimal TSP tour on \( S \cup T \). If \( \sigma > c \), then the constraint

\begin{equation}
y_i + y_j \leq 1 \quad (c_{ii} + c_{ij} + c_{jj} > \sigma) \tag{10}
\end{equation}

is valid.
\[ \sum_{i \in V} y_k \leq |S| - 1 \]  

(10)

is a valid inequality for (STSP).

In the following propositions, denote by \( \bar{x} \) an upper bound on the optimal STSP objective value \( z^* \).

**Proposition 3.** The constraint

\[ \sum_{i \in V} p_i y_k \leq \bar{x} \]  

(11)

is a valid inequality for (STSP). Furthermore, if \( \sum_{i \in S} p_i > \bar{x} \), then the constraint

\[ \sum_{i \in S} y_k \leq |S| - 1 \]  

(12)

is a valid inequality for (STSP).

**Proposition 4.** If \( T \subset S \) and \( \sum_{i \in S} p_i < \bar{x} \), then the constraint

\[ \sum_{i \in V} p_i y_k - \frac{1}{2} (\bar{x} - \sum_{i \in S} p_i) \sum_{i \in V} x_{ij} \leq \sum_{i \in S} p_i \]  

(13)

is a valid inequality for (STSP).

**Proof.** If at the optimum \( \sum_{i \in S, j \in V \setminus S} x_{ij} = 0 \), then \( \sum_{i \in S} p_i \) is an obvious upper bound on the optimal solution value \( \sum_{i \in V} p_i y_k \). If \( \sum_{i \in S, j \in V \setminus S} x_{ij} \geq 2 \), since \( \bar{x} - \sum_{i \in S} p_i > 0 \), the left-hand side of (13) does not exceed \( \sum_{i \in V} p_i y_k - (\bar{x} - \sum_{i \in S} p_i) \) and the constraint holds.

Note that constraint (13) is of no interest if \( T \setminus S \neq \emptyset \) or if \( \sum_{i \in S} p_i \geq \bar{x} \), since it is then dominated by the constraints \( \sum_{i \in S, j \in V \setminus S} x_{ij} = 2 \) or \( \sum_{i \in V} p_i y_k \leq \bar{x} \), respectively.

**Proposition 5.** The constraints

\[ \sum_{i \in V} x_{ij} \leq \sum_{i \in S \setminus \{v_i\}} y_k \]  

(14)

\( (S \subset V, 2 \leq |S| \leq n - 2, \ T \setminus S \neq \emptyset, \ v_i \in S) \)

are valid inequalities for (STSP).

**Proof.** When (3) holds, (14) is algebraically equivalent to (4) since \( 2 \sum_{i \in V} x_{ij} + \sum_{i \in S, j \in V \setminus S \text{ or } i \in S, j \in V \setminus S} x_{ij} = 2y_i + 2 \sum_{i \in S \setminus \{v_i\}} y_k \).

Note that constraints (8) are a special case of (14) whenever \( v_i \in V \setminus T \) and \( v_2 \in V \setminus T \). Also, observe that summing up both sides of (14) over all \( v_i \in S \) yields

\[ |S| \sum_{i \in V \setminus S} x_{ij} \leq (|S| - 1) \sum_{i \in S \setminus \{v_i\}} y_k \]  

(15)

Using the same identity as in the proof of Proposition 5, this is shown to be equivalent to the following connectivity constraint used in Laporte and Martello [17]:

\[ 2 \sum_{i \in V} y_k \leq |S| \sum_{i \in V \setminus S, j \in V \setminus S} x_{ij} \]  

(16)

A graphical interpretation of constraint (16) was provided in Laporte [16]. In the following proposition, \( \bar{z} \) is a lower bound on the optimal profit \( z^* \). In Section 3, we provide a heuristic for determining \( \bar{z} \). Constraints (4) and (14) can sometimes be reinforced as follows:

**Proposition 6.** The connectivity constraints

\[ \sum_{i \in V} x_{ij} \geq 2 \quad (S \subset V, 2 \leq |S| \leq n - 2, \ T \setminus S \neq \emptyset, \ v_i \in S) \]  

(17)

are valid inequalities for (STSP).

**Proof.** This result follows immediately from the fact that the conditions imposed on \( S \) imply the presence of at least two edges between \( S \) and \( V \setminus S \) at the optimum. Indeed, since \( T \setminus S \neq \emptyset \), there must be a connection between \( S \) and \( V \setminus S \) whenever \( S \) contains some compulsory vertices or when a profit at least equal to \( \bar{z} \) cannot be obtained in \( V \setminus S \) alone.

Note that constraints (17) are algebraically equivalent to the subtour elimination constraints

\[ \sum_{i \in V} x_{ij} \leq \sum_{i \in V \setminus S, j \in V \setminus S} x_{ij} \]  

under the same conditions and that (8) is a special case of (18) whenever \( v_i \in T \) and \( v_2 \in V \setminus T \), or \( v_i \in T \) and \( v_2 \in V \setminus T \).

As in the *Covering Tour Problem* [9], 2-matching constraints valid for the TSP (see, e.g., [20]) can be strengthened as follows:
Proposition 7. The constraints

\[ \sum_{v_i \in S} x_{ij} + \sum_{(v_i,v_j) \in E'} x_{ij} = \sum_{v_i \in S} y_i + \frac{1}{2} (|E'| - 1) \]  

for all \( S \subseteq V \) and all \( E' \subseteq E \) satisfying

(i) \(| \{ v_i, v_j \} \cap S | = 1 \) for all \( (v_i,v_j) \in E' \),
(ii) \(| \{ v_i, v_j \} \cap \{ v_k, v_l \} = \emptyset \), \( v_i \neq v_j \neq v_k \neq v_l \) \( \in E' \), and
(iii) \(| E' | \geq 3 \) and odd

are valid inequalities for (STSP).

Proof. Summing up constraints (3) for all \( v_i \in S \), we obtain

\[ 2 \sum_{v_i \in S} x_{ij} + \sum_{v_i \in S, v_j \in E \setminus S} x_{ij} = 2 \sum_{v_i \in S} y_i. \]  

(20)

Since \( x_{ij} = 1 \) for all \( (v_i,v_j) \in E' \), adding these constraints to (20) and dividing by 2 yields

\[ \sum_{v_i \in S} x_{ij} + \sum_{(v_i,v_j) \in E'} x_{ij} = \sum_{v_i \in S} y_i + \frac{1}{2} |E'|. \]  

(21)

As the left-hand side of (21) is integer and \(|E'| \geq 3 \) and is odd, rounding down the right-hand side yields (19). \( \square \)

Finally, if (STSP) is solved by branch-and-cut, the following penalty cut can be applied at a fractional solution obtained at the optimum of a subproblem and all its descendants (see [18]).

Proposition 8. Let \( \xi^* \) be the value taken by a variable \( \xi \{ v_i \} \) or \( x_{ij} \) in (STSP) \) at the optimum of a subproblem. If \( \xi^* \) is fractional, let \( \xi^*_D \) and \( \xi^*_U \) be upper bounds on \( \xi^* \) for the down and up branches \( \xi = 0 \) and \( \xi = 1 \), respectively. Then, the constraint

\[ z \leq \xi^*_D + (\xi^*_U - \xi^*_D) \xi \]  

(22)

is a valid inequality for (STSP).

Proof. The validity of this result follows from the fact that the value of \( z \) must lie on or below the straight line passing through \((0, \xi^*_D)\) and \((1, \xi^*_U)\) in the \( \xi - z \) plane. \( \square \)

The upper bounds \( \xi^*_D \) and \( \xi^*_U \) are computed as in standard sensitivity analysis in linear programming. If the profits are integer, as is the case in our test problems, then \( \xi^*_D \) and \( \xi^*_U \) can be replaced by \( \lfloor \xi^*_D \rfloor \) and \( \lceil \xi^*_U \rceil \) in (22).

3. HEURISTICS

We developed two new heuristics for the STSP. Both make use of the GENIUS composite heuristic for the TSP (Gendreau et al. [8]). Briefly, GENIUS first constructs a TSP tour by inserting, in turn, each vertex \( v \) between two of its \( p \) closest neighbors on the partially constructed tour. At each step, several local reoptimizations of the tour are considered in the area where \( v \) is to be inserted, and the best rearrangement is selected. This construction procedure is followed by a postoptimization phase in which each vertex \( v \) is, in turn, removed from the tour which is then locally reoptimized. Vertex \( v \) is then inserted as above between two of its closest neighbors, until no further improvement is possible.

3.1. Heuristic H1

In this heuristic, an STSP solution is gradually constructed by inserting a pair of vertices, or a single vertex, into the current tour. Periodic reoptimizations of the tour are performed using GENIUS. Once a solution has been determined, an attempt is made to improve it by removing, in turn, two of its vertices, and proceeding as in the construction phase.

Step 1 (Initialization).

Let \( H \) be the set of vertices belonging to the current solution. Let \( z \) be the value of the current feasible solution, and \( z^* \), the value of the best known solution. Set \( H := T \). Determine a TSP tour of cost \( c(H) \) on \( H \) by means of GENIUS. If \( c(H) > c, H1 \) cannot identify a feasible solution and terminates with \( z^* = 0 \). Set \( z := \sum_{v_i \in H} p_i \) and \( z^* := \tilde{z} \). Set \( \delta := 1 \). For every \( v_i \in V \setminus T \), determine its neighborhood \( V_i \), defined as the set of its \( \lambda \) closest neighbors in \( V \setminus T \), where \( \lambda = \min \{| \log n |, | V \setminus T | - 1 | \} \) and \( |x| \) is the value of \( x \) rounded to the nearest integer. This choice of \( \lambda \) ensures that the number of neighbors considered never becomes too large but has a tendency to grow with problem size.

Step 2 (Double insertions).

If \(|V \setminus H| \leq 1 \), go to Step 3. Otherwise, determine \( v_i \in V \setminus H \), \( v_j \in V \setminus H \), and two consecutive vertices \( u_j, u_k \) belonging to the current tour on \( H \) such that \( c(H) + c_{ij} + c_{kl} - c_{jk} \leq c \) and \( (p_i + p_j)/l(c_{ij} + c_{kl}) \) is maximized. (If \(|H| = 1 \), then \( v_j \) and \( v_k \) coincide.) If no such double insertions exists, go to Step 3. Otherwise, update the solution: Set \( H := H \cup \{ v_i, v_j \} \), \( c(H) := c(H) + c_{ij} + c_{kl} - c_{jk} \), \( \tilde{z} := \tilde{z} + p_i + p_j \), and replace \( (v_j, v_k) \) by \( (v_i, v_j, v_k) \) on the current tour. Repeat this step.

Step 3 (Single insertions).

If \(|V \setminus H| = 0 \), the current solution is optimal: Since
all vertices have been included in the tour, stop. Otherwise, determine \( v_i \in V \setminus H \) and two consecutive vertices \( v_j, v_k \in H \) such that \( c(H) + c_{jk} + c_{ik} - c_{jk} \leq c \) and \( p_i / (c_{ji} + c_{ak} - c_{jk}) \) is maximized. If no such insertion is feasible, go to Step 4. Otherwise, update the solution: Set \( H := H \cup \{v_i\} \), \( c(H) := c(H) + c_{ij} + c_{ik} - c_{jk} \), \( z := z + p_i \), replace \((v_j, v_k)\) by \((v_j, v_i, v_k)\) on the current tour, and repeat this step.

**STEP 4** (Reoptimization).
Apply GENIUS to the vertices of \( H \). If this result in a shorter tour, record it, as well as its cost \( c(H) \) and go to Step 2.

**STEP 5** (Incumbent update).
If \( z > z^* \), set \( z^* := z \), record the current solution as the best known, and set \( \delta := 1 \).

**STEP 6** (Termination check).
If \( \delta = 0 \) or \( |H| = 1 \), stop. Otherwise, compute \( H^\prime \), the set of the \( \theta \) vertices of \( H \) having the smallest profits, where \( \theta = \min \{|\log n|, |H|\} \), and set \( \delta := 0 \).
Repeat Step 7 for all pairs \((v_i, v_j)\) where \( v_i \in H \setminus T \) and \( v_j \in H^\prime \setminus T \), \( i \neq j \). Then go to Step 6.

**STEP 7** (Vertex swaps).
Set \( H := H \setminus \{v_i, v_j\} \), \( z := z - p_i - p_j \), and apply GENIUS to \( H \) to obtain a tour of cost \( c(H) \). Execute Steps 2–5.

### 3.2. Heuristic H2
This heuristic constructs a first tour having a cost not exceeding \( 0.8c \), first using all vertices and gradually removing some of them. Vertex insertions are then performed as in H1. An attempt to improve the current solution is then made, by swapping vertices between the current vertex set and its complement. In what follows, we use the same notation as in H1.

**STEP 1** (Initialization).
Set \( H := V \). Determine a TSP tour of cost \( c(H) \) on \( H \) by means of GENIUS. Let \( z := \sum_{e \in H} p_e \).

**STEP 2** (Vertex removals).
Remove from the tour a vertex \( v_i \in H \setminus T \) and maximizing \( (c_{ji} + c_{ak} - c_{jk})/p_i \), where \((v_j, v_i, v_k)\) is a chain of three vertices on the tour. Set \( H := H \setminus \{v_i\} \), \( z := z - p_i \), \( c(H) := c(H) + c_{ji} - c_{ak} + c_{jk} \), and update the tour by connecting \( v_j \) to \( v_k \). If \( c(H) > 0.8c \) and \( H \neq T \), repeat this step.

**STEP 3** (Vertex insertions).
Apply Steps 2–4 of H1, set \( z^* := z \), and record the current solution as the best known.

**STEP 4** (Vertex swaps).
Determine \( v_i \in V \setminus H \) and \( v_j \in H \setminus T \) such that \( c(H \cup \{v_i\} \setminus \{v_j\}) \leq c \) and \( p_i - p_j \) is positive and maximized. Here, the value of \( c(H \cup \{v_i\} \setminus \{v_j\}) \) is determined using GENIUS. If such vertices can be identified, update the best-known solution and repeat this step. Otherwise, stop.

The complexity of these heuristics is \( n \) times that of GENIUS. Each tour improvement step of GENIUS requires \( O(np^2 + n^2) \) operations, where \( p \) is the neighborhood size, but the number of applications of this step is not a polynomial function of \( n \).

### 4. BRANCH-AND-CUT ALGORITHM

We developed a branch-and-cut algorithm for the STSP. This algorithm operates on \((STSP')\), a modified version of \((STSP)\) in which \((1)\) is replaced by a different objective \((1')\) defined as

\[
\text{Maximize } z' = M \sum_{v_i \in V} p_i y_i - \sum_{i < j} c_{ij} x_{ij} \quad (1')
\]

Provided \( M \) is sufficiently large, it is valid to use \((1')\). The use of a modified objective is justified by the fact that the optimal value of the linear relaxation of \((STSP')\) tends to have fewer fractional \( x_i \) variables than that of \((STSP)\) as these variables are now included in the objective. Indeed, for a given value of \( \sum_{v_i \in V} p_i y_i \), the remaining term of \((1')\) coincides with the objective of the TSP, a problem in which its frequent to observe several \( x_i \) variables at integer values. It should be stressed, however, that \((1')\) is used only for computational purposes. The true profit function \( z \) is updated throughout the search tree and fathoming decisions are made by comparing its value to the best-known profit \( z^* \). Initially, we solve a relaxed version of \((STSP')\) including constraints \((2), (3), (6), \) the bounds on the variables, as well as some connectivity constraints of type \((17)\) since these constraints are often binding. At a generic node of the search tree, an attempt to obtain an improved incumbent is made by applying heuristic H1 to \( \{v_i : v_i \in V, y_i \approx 0.3\} \). If this fails to fathom the node, a search for violated constraints among \((4), (8)-(14), (17)-(19), \) and \((22)\) is performed, and some of these constraints are introduced into the current subproblem. This process is reapplied until a feasible or dominated solution is obtained or until it becomes more promising to initiate branching. Periodically, some ineffective constraints are deleted from the program in order to save time and space. All linear programs are solved using the CPLEX code [4].

We now provide a step-by-step description of the algorithm, followed by some implementation details.
STEP 1 (First node of the search tree).
Eliminate all vertices \( v_\ell \) for which 
\( 2c_{1k} > \epsilon \) and compute a lower bound \( z^* \) on \( z^* \) by successively applying heuristics H1 and H2, or only H1, as explained in Section 5. Let \( \theta \) be the number of feasible solutions identified by these heuristics. Among these, only retain all \( \theta' \) solutions having a maximal vertex set with respect to inclusion, and let \( \rho = \min \{ \theta', 30 \} \). Let \( S \) be the vertex set corresponding to solution \( z \) \( \neq z^* \) and go to Step 5; otherwise, apply H1 to \( \{ v_\ell : y_\ell \approx 0.3 \} \) and let \( z_1 \) be the corresponding profit. If \( z_1 > z^* \), set 
\[ z^* := z_1. \]
If \( z^* = z \), go to Step 5.

STEP 2 (Subproblem solution).
Solve subproblem \( \{ P \} \) using CPLEX. Let \( z \) be the associated profit. If \( z \leq z^* \), go to Step 5. If \( z > z^* \), and the solution is feasible or all \( y_\ell \) are integer and the length of the tour produced by GENIUS on \( \{ v_\ell : y_\ell = 1 \} \) is less or equal to \( c \), then set \( z^* := z \) and go to Step 5; otherwise, apply H1 to \( \{ v_\ell : y_\ell \approx 0.3 \} \) and let \( z_1 \) be the corresponding profit. If \( z_1 > z^* \), set 
\[ z^* := z_1. \]
If \( z^* = z \), go to Step 5.

STEP 3 (Redundant constraints elimination).
Among all constraints generated after Step 1, eliminate those that have been ineffective for five applications of Step 4.

STEP 4 (Constraint generation).
Identify up to 50 violated constraints using the following priority list: (11), (8), (12), (9) (if \( |T| = 1 \) ), (10) (if \( |T| > 1 \) ), (13), (4), (17), (14), (18), (19), and (22). This list gives higher priority to the constraints most likely to be binding and easier to identify. As shown below, a better incumbent may be identified while generating constraints (13) or (14). If so, update the incumbent and go to Step 5. If no constraints can be generated, set \( z(\{ P \}) := z \); if \( P = P_1 \), go to Step 6; if \( P = P_0 \), set \( P := P_1 \) and go to Step 2. If some violated constraints can be identified, generate the most violated constraints, up to 30, and go to Step 2.

STEP 5 (Subproblem fathoming).
Set \( z(\{ P \}) := -\infty \). If \( P = P_0 \), set \( P := P_1 \) and go to Step 2.

STEP 6 (Insertion in the list).
If \( z(\{ P \}) = z(\{ P \}) = -\infty \), go to Step 7. If \( z(\{ P \}) > 0 \) and \( z(\{ P \}) > 0 \), insert \( P_0 \) and \( P_1 \) in the list, starting with the subproblem \( \{ P \} \) having the smallest value \( z(\{ P \}) \). If \( z(\{ P \}) = -\infty \) or \( z(\{ P \}) = -\infty \), insert in the list the subproblem \( \{ P \} \) with \( z(\{ P \}) > 0 \).

STEP 7 (Termination check).
If the list is empty, stop. Otherwise, select the last subproblem from the list.

STEP 8 (Branching).
Create two subproblems \( \{ P_0 \} \) and \( \{ P_1 \} \) by branching on a fractional variable in the 0 and 1 direction, respectively. Set \( P := P_0 \) and go to Step 2.

In this algorithm, branching (Step 8) is done in priority on the \( y_\ell \) variables. To determine on which variable to branch first, compute 
\[ y' = \min_{y_\ell \approx 0.5} \{ y_\ell \} \] and 
\[ y'' = \min_{y_\ell \approx 0.5} \{ y_\ell \} \]. Set 
\[ Y = \{ y : y'/2 \leq y \leq y'' + y'/2 \}, \]
and branch on the variable \( y_\ell \in Y \) having the largest profit \( p_\ell \). If all \( y_\ell \) variables are integer but some \( x_\ell \) is fractional, the same rules are applied, but the choice of the branching variable \( x_\ell \) is made according to \( \max \{ c_k \} \).

In Step 4, the identification of violated constraints (8), (9), and (11) is straightforward. As will be seen in the following section, constraints (9) are not always generated. To generate constraints (10), vertices \( v_\ell \in S \cap T \) are sorted in decreasing order of their current \( y_\ell \) value. An attempt to construct a subset \( S \) of \( V \setminus T \) violating (10) is then made. Given \( S \) and \( T, \sigma \) is computed as the length of a shortest spanning 1-tree on \( S \cup T \). A first subset \( S \) is obtained by taking the first two vertices \( v_\ell \) (after sorting) of \( V \setminus S \). The next vertex of \( V \setminus (T \cup S) \) is added to \( S \) until a violation of (10) is found. This process may fail to identify a violated constraint. If it succeeds, a constraint is generated and an attempt to find another violation is made: Let \( v_\ell \) be the last vertex introduced into \( S \), set \( S := S \setminus \{ v_\ell \} \), and consider the next vertex of \( V \setminus (T \cup S) \).

Constraints (12) are generated as follows: Let \( y_k \) denote the value of \( y_k \) in the current subproblem. Then, we wish to determine a subset \( S \) of vertices such that 
\[ \sum_{v_\ell \in S} y_\ell > |S| - 1 \] and 
\[ \sum_{v_\ell \in S} p_\ell u_\ell > \sigma. \] This amounts to solving the Knapsack Problem:

\[
\text{minimize } \sum_{v_\ell \in S} (1 - y_\ell) u_\ell \tag{23}
\]
subject to
\[
\sum_{v_\ell \in S} p_\ell u_\ell \geq \sigma + 1 \tag{24}
\]
\[ u_\ell \in \{ 0, 1 \} \quad (v_\ell \in V \setminus T). \tag{25} \]

Then, \( S = \{ v_\ell : u_\ell = 1 \} \). In practice, (KP) is not solved to optimality, but a greedy heuristic is used to determine an upper bound on its value. Note that Crowder et al. [5] also used this procedure for the separation of cover inequalities.

To identify violated connectivity constraints of type (13), (4), (17), (14), and (18), we proceed as in Fischetti et al. [6]. First, construct a minimum spanning tree on \( G \) with edge weights \( x_\ell \). Whenever an edge is
introduced, let $S$ be the vertex set of the connected component to which the last introduced edge belongs. If $T \subseteq S$ and $\Sigma_{i \in S} p_k = z$, apply GENIUS tour to the vertices of $S$. If the length of the resulting TSP tour on $S$ does not exceed $c$, a new incumbent of profit $z$ has been identified and the constraint generation procedure is aborted. Otherwise, check for constraint violations relative to $S$. When a full spanning tree has been constructed, remove, in turn, one edge at a time and successively set $S$ equal to the vertex sets in each of the two components.

Violated comb inequalities (19) are identified as in Padberg and Rinaldi [20]. Let $G'$ be the subgraph of $G$ induced by the fractional $x_i$ variables. Identify all blocks of $G'$ and use each of them in turn as for the set $S$. Then, attempt to identify a constraint violation by considering all edges with one vertex in $S$ and one vertex in $V \backslash S$.

Finally, penalty cuts (22) are only applied at the root of the search tree, to small-size problems, and on variables $y_k$. More specifically, they are not generated if more than 40 vertices remain after Step 1. When these cuts are generated, we select the $y_k$ variables using the same priority order as for the branching rule.

### 5. Computational Results

The branch-and-cut algorithm just described was coded in C and run on a Sun Sparcstation 1000. As explained below, two types of instances were generated. A time limit of 1200 seconds was imposed on the heuristics and a maximum of 10,000 seconds was allowed for the solution of any instance, including the heuristics. Instances that could not be solved within this limit were deemed unsuccessful.

#### 5.1. Type 1 Instances

In the first set of instances, vertices $v_i$, $i = 1, \ldots, n$, were generated in the $[0, 100]^2$ square according to a continuous uniform distribution. The value of each $c_{ij}$ was then set equal to the Euclidean distance between $v_i$ and $v_j$. An integer profit $p_i$ was assigned to each $v_i \in V \backslash T$ according to a discrete uniform distribution on $[1, 100]$. In a first set of test problems, we used $T = \{v_1\}$, and $c = \beta \tau^*$, where $\beta$ is a parameter and $\tau^*$ is the length of an optimal TSP tour over all vertices, computed by means of the Padberg–Rinaldi algorithm [20]. We also solved instances containing more than one compulsory vertex. Here, we used $|T| = 0.25n$ and $0.50n$, and we declared the first $|T|$ vertices compulsory. To determine $c$ in these instances, we first computed the optimal TSP tour length $\tau^*$ over $T$ and we set $c = \tau^* + \gamma(\tau^* - \tau^*)$, where $\gamma$ is a parameter. For the case $T = \{v_1\}$, we successively set $\beta$ equal to 0.1, 0.3, 0.5, 0.7, and 0.9. For each $\beta$, we gradually increased $n$ by steps of 20, starting with $n = 20$. Five instances were attempted for each combination of $\beta$ and $n$, but we stopped increasing $n$ as soon as all instances for a given problem size were unsuccessful. We then solved five instances using $\gamma = 0.1, 0.3, 0.5, 0.7, 0.9$, and the following combinations of $n$ and $|T|$: $n = 80$, $|T| = 20, 40$, and $n = 100$, $|T| = 25, 50$.

#### 5.2. Type 2 Instances

We also generated instances as in Ramesh et al. [22]. Here, $T = \{v_1\}$. To obtain the $c_{ij}$ coefficients, we first generated costs according to a uniform continuous distribution on $[1, 50]$ and then computed shortest distances. Profits $p_i$ were generated according to a discrete uniform distribution on $[1, 200]$. The value of $c$ was determined as in Type 1 instances with the following exception: When $n$ became too large ($n \geq 200$), the TSP could not be solved optimally and $\tau^*$ had to be replaced by a tour value obtained by means of the GENIUS heuristic [6]. Laporte and Martello [17] also used the same generation mode, but with different parameter values.

#### 5.3. Computational Simplifications

For both types of instances, to reduce computation times when $T = \{v_1\}$, only H1 was applied in Step 1 of the branch-and-cut algorithm whenever $\beta n > 50$. In all other cases, both H1 and H2 were applied. In addition, constraints (9) and (10) were never generated for $\beta > 0.3$ and $|T| = 1$, or $|T| > 1$ respectively.

#### 5.4. Analysis

We report in Tables I–III average computational results over all successful instances. The meanings of the column headings not yet explained are as follows:

- $N$: number of successful instances out of five;
- Number of constraints of each type generated in the course of the branch-and-cut algorithm: In the case of constraint (11), only one constraint is in fact generated, but its right-hand side is updated throughout the algorithm. The reported statistic is one plus the number of such updates;
- LB/OPT: heuristic solution value divided by the optimal solution value;
- UB/OPT: upper-bound value computed at the root of the search tree, divided by the optimal solution value;
- NODES: number of nodes generated in the branch-and-cut tree;
- TOUR SIZE: number of vertices included in the optimal STSP solution;
that our algorithm solved undirected instances. As instances become more difficult to solve as they tend to

\[ T \]

value of \( T \) grows, results presented in Table I for Type 1 instances indicate that our algorithm solved undirected instances involving between 100 and 300 vertices, depending on the value of \( \beta \). As experienced by Laporte and Martello [17], problem difficulty increases with \( \beta \). This is easily explained by the fact that instances with a low value of \( \beta \) (and thus a low limit \( c \)) are more constrained and lead to early fathoming of branches in the search tree. In particular, several constraints of types (9), (11), and (12) can be imposed near the root of the search tree. As \( \beta \) grows, instances become more difficult to solve as they tend to resemble a pure TSP and the vertex sequencing aspect becomes more predominant.

<table>
<thead>
<tr>
<th>( \beta )</th>
<th>( n )</th>
<th>( N )</th>
<th>( (11) )</th>
<th>( (8) )</th>
<th>( (12) )</th>
<th>( (9) )</th>
<th>( (13) )</th>
<th>( (4, 14, 17, 18) )</th>
<th>( (19) )</th>
<th>( (22) )</th>
<th>( LB/OPT )</th>
<th>( UB/OPT )</th>
<th>NODES</th>
<th>TOUR SIZE</th>
<th>CPU H</th>
<th>CPU BC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>20</td>
<td>5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.000</td>
<td>1.000</td>
<td>2.4</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>0.8</td>
<td>5</td>
<td>8.2</td>
<td>11.6</td>
<td>15.6</td>
<td>18.6</td>
<td>21.6</td>
<td>24.6</td>
<td>27.6</td>
<td>30.6</td>
<td>0.968</td>
<td>1.005</td>
<td>4.8</td>
<td>6.8</td>
<td>1</td>
<td>2.2</td>
</tr>
<tr>
<td>60</td>
<td>0.0</td>
<td>16.0</td>
<td>30.0</td>
<td>44.4</td>
<td>54.6</td>
<td>73.8</td>
<td>11.4</td>
<td>14.1</td>
<td>17.1</td>
<td>20.1</td>
<td>0.995</td>
<td>1.030</td>
<td>9.4</td>
<td>2.2</td>
<td>5.6</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>1.0</td>
<td>0.0</td>
<td>5.6</td>
<td>8.0</td>
<td>11.0</td>
<td>14.0</td>
<td>17.0</td>
<td>20.0</td>
<td>23.0</td>
<td>26.0</td>
<td>0.986</td>
<td>1.031</td>
<td>5.8</td>
<td>11.8</td>
<td>7.4</td>
<td>40.2</td>
</tr>
<tr>
<td>120</td>
<td>1.4</td>
<td>16.0</td>
<td>8.6</td>
<td>30.0</td>
<td>119.0</td>
<td>20.2</td>
<td>40.0</td>
<td>1.000</td>
<td>1.005</td>
<td>4.8</td>
<td>1.2</td>
<td>19.2</td>
<td>38.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>1.0</td>
<td>26.0</td>
<td>40.0</td>
<td>30.0</td>
<td>119.0</td>
<td>20.0</td>
<td>40.0</td>
<td>1.040</td>
<td>1.060</td>
<td>15.8</td>
<td>44.6</td>
<td>64.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>5.4</td>
<td>45.0</td>
<td>12.4</td>
<td>20.4</td>
<td>119.0</td>
<td>20.0</td>
<td>40.0</td>
<td>1.020</td>
<td>1.040</td>
<td>5.0</td>
<td>19.6</td>
<td>81.8</td>
<td>26.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>5.3</td>
<td>53.0</td>
<td>14.6</td>
<td>20.8</td>
<td>119.0</td>
<td>20.0</td>
<td>40.0</td>
<td>1.080</td>
<td>1.100</td>
<td>5.0</td>
<td>22.0</td>
<td>134.4</td>
<td>421.8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>5.0</td>
<td>101.6</td>
<td>230.4</td>
<td>102.4</td>
<td>119.0</td>
<td>20.0</td>
<td>40.0</td>
<td>1.090</td>
<td>1.110</td>
<td>23.0</td>
<td>190.6</td>
<td>2390.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>220</td>
<td>7.9</td>
<td>199.0</td>
<td>106.0</td>
<td>302.6</td>
<td>119.0</td>
<td>20.0</td>
<td>40.0</td>
<td>1.090</td>
<td>1.110</td>
<td>26.0</td>
<td>26.4</td>
<td>317.6</td>
<td>1574.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>240</td>
<td>9.7</td>
<td>245.8</td>
<td>134.0</td>
<td>390.2</td>
<td>119.0</td>
<td>20.0</td>
<td>40.0</td>
<td>1.090</td>
<td>1.110</td>
<td>27.0</td>
<td>28.8</td>
<td>410.6</td>
<td>2999.6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>260</td>
<td>10.5</td>
<td>235.0</td>
<td>113.0</td>
<td>461.3</td>
<td>119.0</td>
<td>20.0</td>
<td>40.0</td>
<td>1.090</td>
<td>1.110</td>
<td>30.0</td>
<td>31.7</td>
<td>728.0</td>
<td>3890.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>280</td>
<td>12.0</td>
<td>278.5</td>
<td>3.0</td>
<td>508.5</td>
<td>1096.5</td>
<td>1205.0</td>
<td>6.0</td>
<td>0.0</td>
<td>1.080</td>
<td>1.100</td>
<td>9.0</td>
<td>35.0</td>
<td>927.5</td>
<td>4422.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>8.8</td>
<td>284.0</td>
<td>168.0</td>
<td>386.0</td>
<td>999.0</td>
<td>806.0</td>
<td>15.0</td>
<td>0.0</td>
<td>0.880</td>
<td>1.004</td>
<td>99.0</td>
<td>36.0</td>
<td>1200.0</td>
<td>6147.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CPU H: number of CPU seconds required to run the heuristic(s);

CPU BC: number of CPU seconds required to run the branch-and-cut algorithm, excluding the two heuristics.

Results presented in Table I for Type 1 instances indicate that our algorithm solved undirected instances involving between 100 and 300 vertices, depending on the value of \( \beta \). As experienced by Laporte and Martello [17], problem difficulty increases with \( \beta \). This is easily explained by the fact that instances with a low value of \( \beta \) (and thus a low limit \( c \)) are more constrained and lead to early fathoming of branches in the search tree. In particular, several constraints of types (9), (11), and (12) can be imposed near the root of the search tree. As \( \beta \) grows, instances become more difficult to solve as they tend to resemble a pure TSP and the vertex sequencing aspect becomes more predominant.
results in more than 50% of all vertices being present in the solution, thus yielding a much more difficult problem. In fact, the STSP with $|T|$ compulsory vertices is harder in practice than is a TSP with $|T|$ vertices since one must determine which vertices belong to the solution, and an optimal sequence must be computed typically over far more than $|T|$ vertices.

Results presented in Table III indicate that Type 2 instances are much easier to solve than are Type 1 instances and that larger problem sizes, involving up to 300 vertices, can be solved to optimality even for very large values of $\beta$. This exceeds by far the largest size attained by Laporte and Martello [17] ($n = 90$) and by Ramesh et al. [22] ($n = 150$) who tested their algorithms on comparably small instances. Taking computer types into account, our algorithm would appear to be slower than that of Ramesh et al. when $n \leq 150$. However, the gap ($UB / OPT$) at the root of the search tree is much smaller: It is rather difficult. Finally, the application of $H_1$ in Step 3 of the branch-and-cut algorithm helped generate a better incumbent in a large number of cases.

Results presented in Table II for $|T| = 0.25n$ and $0.50n$ show that our algorithm also performs very well on this type of instance. Comparisons of these results with the case $T = \{ v_i \}$ must be made with care. In particular, the parameters $\beta$ and $\gamma$ do not have the same impact on the solution. For example, setting $\beta = 0.1$ results in incorporating approximately 10% of all vertices in the solution, leading to a relatively easy problem for the branch-and-cut algorithm. In contrast, setting $\gamma = 0.1$...
by computing shortest paths, dominance relationships often occur: Whenever \( c_{ij} = c_{ik} + c_{kj} \) for some \( i, j, k \), the branch \( \langle x_{ij} = 1 \rangle \) is dominated.

6. CONCLUSION

We have developed several new classes of valid inequalities, as well as two heuristics and a branch-and-cut algorithm for the symmetric STSP. These algorithms were tested on two classes of problems ranging from 20 to 300 vertices. Our results show that problem difficulty is largely dependent on the tightness factor \( \beta \) and on the generation mode. Tighter problems tend to be easier since the optimal solution contains fewer vertices. Instances with costs generated from a random distribution and shortest-path computations are easier than those with vertices randomly generated on the plane. Depending on problem type, instances involving up to 300 vertices can be solved to optimality by means of our branch-and-cut algorithm. This by far exceeds the size of problems that can be tackled using alternative approaches.

This work was partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) under Grants OGP0038816 and OGP0039682. F.S. also benefited from an NSERC International postdoctoral fellowship and was partially funded by Strategic Grant STR0149269. Thanks are also due to Manfred W. Padberg and Giovanni Rinaldi who made their TSP code available to the authors and to three anonymous referees for their valuable comments.

REFERENCES


