Solving the Orienteering Problem through Branch-and-Cut

Article *in* Informs Journal on Computing · May 1998

DOI: 10.1287/ijoc.10.2.133 · Source: DBLP

3 authors:

Matteo Fischetti
University of Padova
212 PUBLICATIONS **7,933** CITATIONS

SEE PROFILE

Juan José Salazar González
Universidad de La Laguna
164 PUBLICATIONS **3,400** CITATIONS

SEE PROFILE

Paolo Toth
University of Bologna
236 PUBLICATIONS **17,791** CITATIONS

SEE PROFILE

INFORMS Journal on Computing
Vol. 10, No. 2, Spring 1998

0899-1499/98/0902-0133 $05.00

Solving the Orienteering Problem through Branch-and-Cut

MATTEO FISCHETTI / *DEI, University of Padova, 35131 Padova, Italy, Email: fisch@dei.unipd.it*

JUAN JOSÉ SALAZAR GONZÁLEZ / *DEIOC, University of La Laguna, 38271 Tenerife, Spain, Email: jjsalaza@ull.es*

PAOLO TOTH / *DEIS, University of Bologna, 40136 Bologna, Italy, Email: ptoth@deis.unibo.it*

(Received: October 1996; revised: November 1997; accepted: February 1998)

In the Orienteering Problem (OP), we are given an undirected graph with edge weights and node prizes. The problem calls for a simple cycle whose total edge weight does not exceed a given threshold, while visiting a subset of nodes with maximum total prize. This NP-hard problem arises in routing and scheduling applications. We describe a branch-and-cut algorithm for finding an optimal OP solution. The algorithm is based on several families of valid inequalities. We also introduce a family of cuts, called conditional cuts, which can cut off the optimal OP solution, and propose an effective way to use them within the overall branch-and-cut framework. Exact and heuristic separation algorithms are described, as well as heuristic procedures to produce near-optimal OP solutions. An extensive computational analysis on several classes of both real-world and random instances is reported. The algorithm proved to be able to solve to optimality large-scale instances involving up to 500 nodes, within acceptable computing time. This compares favorably with previous published methods.

We consider the following routing problem, related to the well-known Traveling Salesman Problem (TSP). We are given a set of n cities, each having an associated nonnegative prize, and a vehicle stationed in a depot located (say) in city 1. Let $t_{ij} = t_{ji}$ be the time spent for routing cities i and j in sequence. The Orienteering Problem (OP) is to find a route for a vehicle, visiting each city at most once, requiring a total time not exceeding a given bound t_0, and collecting a maximum total prize. This problem is NP-hard, and arises in several routing and scheduling applications, see e.g., Golden, Levy, and Vohra.[12]

Heuristic algorithms for OP and some generalizations have been proposed by Tsiligirides,[25] Golden, Levy, and Vohra,[12] Golden, Wang, and Liu,[13] and Chao, Golden, and Wasil.[6] Exact enumerative methods have been proposed by Laporte and Martello,[15] and by Ramesh, Yoon, and Karwan.[22] Leifer and Rosenwein[16] have discussed an LP-based bounding procedure. Recently, Gendreau, Laporte, and Semet[11] have proposed a branch-and-cut approach.

Related to OP is the Prize Collecting TSP introduced by Balas and Martin[4] as a model for scheduling the daily operations of a steel-rolling mill. In this problem, one has to find a route for a vehicle, visiting each city at most once, collecting a total prize not less than a given bound and having minimum total time. Balas[1, 2] gave polyhedral results for this problem. A branch-and-bound algorithm for its exact solution has been presented in Fischetti and Toth.[10]

Also related to OP is the Cycle Problem, calling for a minimum cost route for a vehicle, visiting each city at most once. If negative cost circuits are allowed, this problem is NP-hard. Balas[3] and Bauer,[5] among others, studied the facial structure of a polytope associated with the problem.

In this article, we propose an exact branch-and-cut algorithm for OP. In Section 1, a basic mathematical model is presented. Section 2 discusses a number of additional constraints that improve the quality of the LP relaxation of the basic model. We also introduce a family of conditional cuts, i.e., cuts that cut off the current optimal solution. Separation procedures are described in Section 3, whereas Section 4 gives a heuristic algorithm for finding approximate OP solutions. The overall branch-and-cut algorithm is described in Section 5. In this section we also propose an effective way of integrating conditional cuts within the overall framework. Extensive computational results on several classes of test problems are presented in Section 6, showing that the approach is capable of solving large-scale OP instances involving up to 500 nodes within reasonable computing times. This compares favorably with previous results from the literature.

An early version of this paper was presented at the XIII Conference of EURO held in Glasgow (U.K.), July 1994.

1. Basic Mathematical Model

We consider a complete (undirected) graph $G = (V, E)$ with $n := |V|$ nodes. Node 1 represents the depot. Let p_v denote the nonnegative prize associated with each $v \in V$ (with $p_1 = 0$), let t_e be the nonnegative travel time associated with any $e \in E$, and let t_0 be the maximum total travel time allowed for the vehicle.

We assume throughout that all values p_v, t_e, and t_0 are integer. Moreover, we assume without loss of generality, that OP has at least one feasible solution (this can be checked in polynomial time, by finding the shortest cycle through node 1).

For $S \subset V$ we define

$$E(S) := \{[u, v] \in E: u \in S, v \in S\},$$

$$\delta(S) := \{[u, v] \in E: u \in S, v \notin S\},$$

Subject classifications: Orienteering problem
Other key words: Mathematical programming, branch and cut.

and for any $v \in V$ we write $\delta(v)$ instead of $\delta(\{v\})$. Moreover, for any given $T \subseteq E$ we define

$$V(T) := \{v \in V: T \cap \delta(v) \neq \emptyset\}$$

as the set of the nodes spanned by T. Given a real function f on a finite domain W and a set $S \subseteq W$, we write $f(S)$ for $\Sigma_{w \in S} f(w)$.

We use two types of decision variables, x_e and y_v, associated with the edges and nodes of G, respectively, with the following meaning:

$$x_e := \begin{cases} 1 & \text{if edge } e \text{ is used,} \\ 0 & \text{otherwise,} \quad e \in E; \end{cases}$$

$$y_v := \begin{cases} 1 & \text{if vertex } v \text{ is visited,} \\ 0 & \text{otherwise,} \quad v \in V. \end{cases}$$

OP can then be formulated as the 0–1 Integer Linear Program:

$$v(\text{OP}) := \max \sum_{v \in V} p_v y_v \tag{1}$$

subject to

$$\sum_{e \in E} t_e x_e \leq t_0, \tag{2}$$

$$x(\delta(v)) = 2y_v \quad \text{for } v \in V, \tag{3}$$

$$x(\delta(S)) \geq 2y_v \quad \text{for } S \subset V, 1 \in S, v \in V \backslash S, \tag{4}$$

$$y_1 = 1, \tag{5}$$

$$0 \leq x_e \leq 1 \quad \text{for } e \in E, \tag{6}$$

$$0 \leq y_v \leq 1 \quad \text{for } v \in V \backslash \{1\}, \tag{7}$$

$$x_e \text{ integer} \quad \text{for } e \in E, \tag{8}$$

$$y_v \text{ integer} \quad \text{for } v \in V \backslash \{1\}. \tag{9}$$

Recall that, for any given $F \subseteq E$, $x(F)$ stands for $\Sigma_{e \in F} x_e$. Constraint (2) imposes the bound t_0 on the total travel time. The degree equations (3) stipulate that a feasible solution has to go exactly once through each visited node. The Generalized Subtour Elimination Constraints (GSECs) (4) force each visited node $v \in V \backslash \{1\}$ to be reachable from node 1 by means of two edge-disjoint paths.

Because of the degree equations (3), the GSECs can equivalently be written as

$$x(E(S)) \leq y(S) - y_v \quad \text{for } S \subset V, 1 \in S, v \in V \backslash S \tag{10}$$

and

$$x(E(\bar{S})) \leq y(\bar{S}) - y_v \quad \text{for } \bar{S} \subset V, 1 \in V \backslash \bar{S}, v \in \bar{S}, \tag{11}$$

where, for any given $Q \subseteq V$, $y(Q)$ stands for $\Sigma_{v \in Q} y_v$. Notice that the inequalities

$$x(\delta(S)) \geq 2(y_i + y_j - 1)$$
$$\text{for } S \subset V, 1 \in S, i \in S, j \in V \backslash S$$

although valid, are dominated by (4) because $y_i - 1 \leq 0$ for all $i \in V$. Finally (5) imposes that node 1 must be visited, and (6)–(9) require that all variables are 0–1 valued. Observe that cycles of length 2 are not allowed in the model. Indeed, an optimal such cycle can easily be found, by enumeration, in $O(n)$ time; hence we can assume, without loss of generality, that the optimal cycle contains at least 3 edges.

2. Additional Inequalities

In this section, we describe the five classes of additional inequalities for OP that we use in our branch-and-cut scheme. These inequalities are redundant as long as the integrality restriction on the variables is retained, but capable of strengthening the LP-relaxation (1)–(7) of the model. The first two classes do not rely on the total time restriction (2), and are derived from the cycle relaxation of OP[3, 5]. The remaining classes, instead, do exploit the total time restriction.

Let P_{OP}, P_{CP}, P_{PCTSP}, and P_{TSP} denote the convex hull of the integer solutions to the OP, the cycle problem, the Prize Collecting TSP, and the TSP, respectively. P_{TSP} is easily seen to be a face of both P_{CP} and P_{PCTSP}, which allows one to extend known results of the facial structure of P_{TSP} to both P_{PC}[3, 5] and P_{PCTSP}[1, 2]. This is not the case of P_{OP}, in that $P_{\text{TSP}} \cap P_{\text{OP}} = \emptyset$ whenever the distance constraint (2) is active. In view of this fact, a polyhedral analysis of whether the proposed inequalities are facet-defining for P_{OP} appears rather difficult, and is not addressed in the present article. Diaby and Ramesh[8] present similar arguments on a related problem. From the practical point of view, however, these cuts proved of fundamental importance for the solution of most instances of our test bed.

Logical Constraints

Clearly, $x_e = 1$ for some $e \in \delta(j)$ implies $y_j = 1$. Hence the logical constraints

$$x_e \leq y_j \quad \text{for all} \quad e \in \delta(j), j \in V \backslash \{1\} \tag{12}$$

are valid for OP. Whenever $e = [v, j] \notin \delta(1)$, inequality (12) is a particular case of (11) arising for $\bar{S} = \{v, j\}$. On the other hand, for $e \in \delta(1)$, these inequalities do improve the LP relaxation of model (1)–(7). To see this, consider the fractional point (x^*, y^*) with $x^*_{12} = x^*_{13} = 1$, $y^*_1 = 1$, $y^*_2 = y^*_3 = \frac{1}{2}$ (all other components being 0). Assuming $t_{12} + t_{13} \leq t_0$, this point satisfies all the constraints of the relaxation, but not the constraints (12) associated with $e = [1, 2]$ and $j = 2$, and with $e = [1, 3]$ and $j = 3$.

We observe that the addition of (12) to model (1)–(9) makes the integrality requirement on the y-variables redundant. Indeed, let (x^*, y^*) be any point satisfying (2)–(8), and define $T^* := \{e \in E: x^*_e = 1\}$. Then from (3), we have $y_v = |T^* \cap \delta(v)|/2$ for all $v \in V$, i.e., $y_v \in \{0, \frac{1}{2}, 1\}$. But $y_v = \frac{1}{2}$ would imply $T^* \cap \delta(v) = \{e\}$ for some $e \in \delta(v)$, which is impossible

because in this case, the corresponding logical constraint (12) would be violated.

2. Matching Inequalities

The well-known 2-Matching Constraints for the TSP have the following counterpart in the cycle relaxation of OP:

$$x(E(H)) + x(T) \leq y(H) + \frac{|T| - 1}{2}, \qquad (13)$$

where $H \subset V$ is called the handle, and $T \subset \delta(H)$ is a set of $|T| \geq 3$, $|T|$ odd, pairwise disjoint teeth. This inequality is obtained by adding up the degree constraints for all $v \in H$ and the bound constraints $x_e \leq 1$ for all $e \in T$, dividing by 2, and then rounding down all the coefficients to the nearest integer.

Cover Inequalities

The total time constraint (2), along with the requirements $x_e \in \{0, 1\}$ for $e \in E$, defines an instance of the 0–1 Knapsack Problem, in which items correspond to edges. Therefore, every valid knapsack problem inequality can be used to hopefully improve the LP relaxation of the OP model. Among the several classes of known knapsack problem inequalities, we have considered the cover inequality (see, e.g., Nemhauser and Wolsey[18]):

$$x(T) \leq |T| - 1, \qquad (14)$$

where $T \subseteq E$ is an inclusion-minimal edge subset with $\Sigma_{e \in T} t_e > t_0$. This constraint stipulates that not all the edges of T can be selected in a feasible OP solution.

A cover inequality can, in some cases, be strengthened. In particular, one can easily obtain the valid extended inequality

$$x(T \cup Q) \leq |T| - 1, \qquad (15)$$

where $Q := \{e \in E \backslash T: t_e \geq \max_{f \in T} t_f\}$.

A different improvement is next proposed, which exploits the fact that the selected edges have to define a cycle. The improvement can only be applied if T defines an infeasible cycle passing through node 1, and leads to the cycle-cover inequality:

$$x(T) \leq y(V(T)) - 1. \qquad (16)$$

Validity of (16) follows from the easy observation that $x(T) \geq y(V(T))$ would imply $x_e = 1$ for all $e \in T$. Figure 1 shows a fractional point violating a cycle-cover inequality, but not other previous inequalities. More generally, (16) is a valid inequality whenever T does not contain any feasible cycle. This generalization will be studied in the forthcoming subsection on conditional cuts.

Path Inequalities

The previous classes of additional inequalities (except the cycle-cover inequalities) are based either on the cycle or on the knapsack relaxation of the problem. We next introduce a new family of constraints that exploit both relaxations.

Let $P = \{[i_1, i_2], [i_2, i_3], \ldots, [i_{k-1}, i_k]\}$ be any simple path through $V(P) = \{i_1, \ldots, i_k\} \subseteq V\backslash\{1\}$, and define the nodeset:

$W(P) := \{v \in V\backslash V(P):$

$P \cup \{[i_k, v]\}$ can be part of a feasible OP solution$\}$.

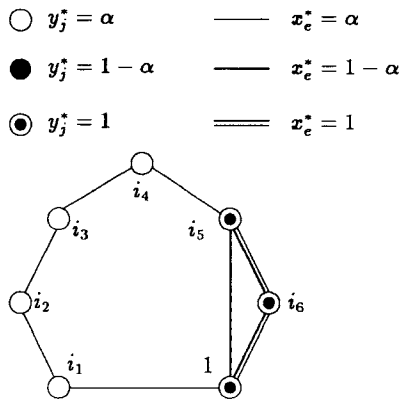

Figure 1. Fractional point violating a cycle cover inequality for the OP instance with $t_0 = 6$ and $t_e = 1$ for all $e \in E$ $(0 < \alpha \leq \frac{1}{2})$. Here $T = \{[1, i_1], [i_1, i_2], \ldots, [i_6, 1]\}$, $x(T) = 2 + 5\alpha$, and $y(V(T)) = 3 + 4\alpha$.

We allow P to be infeasible, in which case $W(P) = \emptyset$. Then the following path inequality

$$\sum_{j=1}^{k-1} x_{i_j i_{j+1}} - \sum_{j=2}^{k-1} y_{i_j} - \sum_{v \in W(P)} x_{i_k v} \leq 0 \qquad (17)$$

is valid for OP. Indeed, suppose there exists a feasible OP solution (x^*, y^*) violating (17). Then

$$x^*_{i_1 i_2} + (x^*_{i_2 i_3} - y^*_{i_2}) + \cdots + (x^*_{i_{k-1} i_k} - y^*_{i_{k-1}}) - \sum_{v \in W(P)} x^*_{i_k v} \geq 1,$$

where $x^*_{i_j i_{j+1}} - y^*_{i_j} \leq 0$ for all $j = 2, \ldots, k-1$. It then follows that $x^*_{i_1 i_2} = 1$ (hence $y^*_{i_2} = 1$), $x^*_{i_2 i_3} - y^*_{i_2} = 0$ (hence $x^*_{i_2 i_3} = 1$ and $y^*_{i_3} = 1$), $\ldots$, $x^*_{i_{k-1} i_k} - y^*_{i_{k-1}} = 0$ (hence $x^*_{i_{k-1} i_k} = 1$), and $x^*_{i_k v} = 0$ for all $v \in W(P)$. But then solution (x^*, y^*) cannot be feasible, because it contains all the edges of P, plus an edge $[i_k, w]$ with $w \notin W(P)$.

Figure 2 shows a typical fractional point that is cut off by a path inequality. This point can be viewed as the convex combination of two cycles, one of which is infeasible because of the total time requirement. It is not difficult to check that the point satisfies all the previous inequalities.

Recall that, for any given $F \subseteq E$, $t(F)$ stands for $\Sigma_{e \in F} t_e$. The definition of $W(P)$ amounts to checking for each $v \in V\backslash V(P)$ whether there exists a cycle of the form $C = P_1 \cup (P \cup \{i_k, v\}) \cup P_2$, where P_1 and P_2 are node-disjoint paths from 1 to i_1 and v, respectively, such that $t(P_1) + t(P) + t_{i_k v} + t(P_2) \leq t_0$. A simpler condition (producing a possibly larger set $W(P)$, and hence a weakened inequality (17)) is obtained by removing the requirement that P_1 and P_2 share no node (except node 1). This leads to the alternative definition of $W(P)$ as

$W(P) := \{v \in V\backslash V(P):$

$$d(1, i_1) + t(P) + t_{i_k v} + d(1, v) \leq t_0\}, \qquad (18)$$

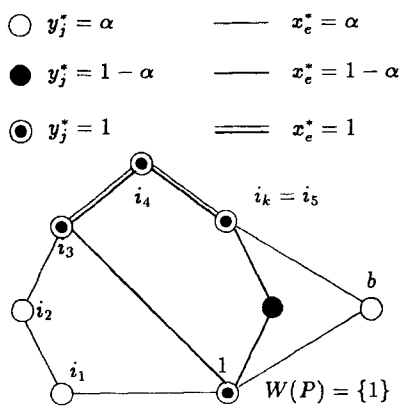

Figure 2. A fractional point violating a path inequality for the OP instance with $t_0 = 6$ and $t_e = 1$ for all $e \in E$ ($0 < \alpha \leq \frac{1}{2}$).

where for each $j \in V\backslash\{1\}$, $d(1, j)$ gives the total time associated with the shortest path from node 1 to node j.

Conditional Cuts

We next address inequalities that are not guaranteed to be valid for our problem, but can nevertheless be used in a cutting-plane context.

Suppose that a heuristic OP solution of value (say) LB is available. In the process of finding an optimal OP solution, we are clearly interested in finding, if any, a feasible solution of value strictly better than LB. Therefore, any inequality can be exploited as a cutting plane, provided that it is satisfied by every feasible OP solution of value greater than LB. We call this kind of inequalities conditional cuts.

We have considered a general family of inequalities of the type

$$x(T) \leq y(V(T)) - 1, \tag{19}$$

where $T \subset E$ is chosen in an appropriate way. It can be seen easily that $x(T) \leq y(V(T))$ holds for every feasible solution, no matter how T is chosen. Moreover, $x(T) = y(V(T))$ implies that the OP solution consists of a cycle entirely contained in T. It then follows that (19) can be used as a conditional cut, provided that no feasible OP solution of value strictly greater than LB is contained in T. This occurs, in particular, when

$$T = E(S) \quad \text{for some } S \subset V$$
$$\text{such that } 1 \in S \text{ and } \sum_{v \in S} p_v \leq LB. \tag{20}$$

A different approach for defining conditional cuts, based on enumeration, will be described in the following section.

3. Separation Algorithms

In this section, we outline exact and/or heuristic algorithms for the following separation problem: Let $\mathcal{F}$ be one of the families of OP inequalities described in Section 2; given a point $(x^*, y^*) \in [0, 1]^{E \times V}$, which satisfies (2)–(3), find a member $\alpha x + \beta y \leq \gamma$ of $\mathcal{F}$ which is (mostly) violated by (x^*, y^*), if any.

We denote by $G^* = (V^*, E^*)$ the support graph associated with the given (x^*, y^*), where $V^* := \{v \in V: y_v^* > 0\}$ and $E^* := \{e \in E: x_e^* > 0\}$.

GSECs

Let x_e^* be viewed as a capacity value associated with each $e \in E^*$. For any fixed $v \in V^*\backslash\{1\}$, we determine a most violated GSEC (among those for the given v) by finding a minimum-capacity $(1, v)$-cut, say $(S_v, V^*\backslash S_v)$, on G^*. This requires $O(|V^*|^3)$ time, in the worst case, through a max-flow algorithm. Trying all possible $v \in V^*\backslash\{1\}$ then leads to an overall $O(|V^*|^4)$-time separation algorithm.

In our implementation, we consider the nodes v in decreasing order of the associated y_v^*. Whenever a violated GSEC is found for (say) the pair S_v and v, we increase the capacity x_{1v}^* by the quantity $2 - x^*(\delta(S_v))$. This prevents the cut $(S_v, V^*\backslash S_v)$ from being generated again in the subsequent iterations. Moreover, to increase the number of violated inequalities detected by a single max-flow computation, we consider two minimum-capacity $(1, v)$-cuts for each v, namely $(S_v', V^*\backslash S_v')$ and $(V^*\backslash S_v, S_v)$, where S_v (respectively, S_v') contains the nodes connected to node v (respectively, node 1) in the incremental graph corresponding to the maximum flow vector. Nodeset S_v gives a hopefully violated GSEC, whereas S_v' is used, as explained later, for producing a conditional cut.

Logical Constraints

This family can be dealt with by complete enumeration, with an overall $O(|E^*|)$ time complexity.

2. Matching Constraints

These inequalities can be separated in polynomial time through a simple modification of the Padberg and Rao[19] odd-cut separation scheme. To reduce the computational effort spent in the separation, however, we have implemented the following simple heuristic. Values x_e^* are interpreted as weights associated with the edges. We apply the greedy algorithm of Kruskal to find a minimum-weight spanning tree on G^*. At each iteration in which this algorithm selects a new edge e, we determine (in the subgraph of G^* induced by all the edges selected so far) the connected component which contains e, say H. The nodeset H is then considered as the handle of a hopefully violated 2-matching constraint. In this way, we generate efficiently all the connected components H of the subgraph $G_\theta = (V, E_\theta)$ induced by $E_\theta := \{e \in E: 0 < x_e^* < \theta\}$ for every possible threshold θ. These sets H have high probability of being the handle of a violated 2-matching constraint, if one exists. For any H, tooth edges are determined, in an optimal way, through the following greedy procedure. Let $\delta(H) = \{e_1, \ldots, e_p\}$ with $x_{e_1}^* \geq x_{e_2}^* \geq \cdots \geq x_{e_p}^*$. We initially relax the requirement that the teeth have to be pairwise disjoint. For any given $|T| \geq 3$ and odd, the best choice for T consists of the edges $e_1, \ldots, e_{|T|}$. Therefore a most-violated inequality corresponds to the choice of the odd integer $|T| \geq 3$ which maximizes

$$x_{e_1}^* + (x_{e_2}^* + x_{e_3}^* - 1) + \cdots + (x_{e_{|T|-1}}^* + x_{e_T}^* - 1).$$

If no violated cut can be produced in this way, then clearly, no violated 2-matching constraint exists for the given handle. Otherwise, we have a violated 2-matching constraint in which two tooth edges, say e and f, may overlap in a node, say v. In this case, we simplify the inequality by defining a new handle–tooth pair (H', T') with $T' := T \backslash \{e, f\}$, and $H' := H \backslash \{v\}$ (if $v \in H$) or $H' := H \cup \{v\}$ (if $v \notin H$). It is then easy to see that the inequality (13) associated with this new pair (H', T') is at least as violated as that associated with the original pair $(H; T)$. Indeed, replacing (H, T) with (H', T') increases the violation by, at least, $1 + y_v - x(\delta(v)) \geqslant 2y_v - x(\delta(v)) = 0$ (if $v \in H$), or $1 - y_v \geqslant 0$ (if $v \notin H$). By iterating this simplification step, one can then always detect a violated 2-matching constraint with nonoverlapping teeth. In some cases, this procedure could even lead to a 2-matching constraint with $|T| = 1$; if this occurs, we reject the inequality in favour of a GSEC associated with the handle.

Path Inequalities

We assume that the fractional point (x^*, y^*) satisfies all the logical constraints (12) as it is the case in our implementation. We observe that the path inequality associated with a given path P cannot be violated by (x^*, y^*) if $x^*_{i_h i_{h+1}} = 0$ for some $[i_h, i_{h+1}] \in P$. This follows from the fact that (17) can be rewritten as

$$\sum_{j=1}^{h-1} (x_{i_j i_{j+1}} - y_{i_{j+1}}) + x_{i_h i_{h+1}} + \sum_{j=h+1}^{k-1} (x_{i_j i_{j+1}} - y_{i_j}) - \sum_{v \in W(P)} x_{i_k v} \leqslant 0$$

where all terms involved in the first two summations are nonpositive by assumption. Hence, every violated path inequality must be associated with a path P contained in the support graph G^*. Because this graph is usually very sparse, we have implemented a simple enumeration scheme to detect the path P producing a most-violated path inequality. We start with an empty node sequence P. Then, iteratively, we extend the current P in any possible way, and check whether the associated path inequality is violated. Whenever, for the current path $P = \{[i_1, i_2], \ldots, [i_{k-1}, i_k]\}$ we have

$$\sum_{j=1}^{k-1} x_{i_j i_{j+1}} - \sum_{j=2}^{k-1} y_{i_j} \leqslant 0,$$

we backtrack, because no extension of P can lead to a violated cut.

Cover Inequalities

We first address the separation problem for cover inequalities in their weakest form (14), which call for an edgeset T with $\Sigma_{e \in T} t_e > t_0$, which maximizes $x^*(T) - |T| + 1$. It is well known that this problem can be formulated as

$$\sigma^* := \min \sum_{e \in E} (1 - x^*_e) z_e \tag{21}$$

$$\text{subject to} \quad \sum_{e \in E} t_e z_e \geqslant t_0 + 1, \tag{22}$$

$$z_e \in \{0, 1\} \quad \text{for } e \in E. \tag{23}$$

Although NP-hard, the knapsack problem (21)–(23) can typically be solved within short computing time by means of specialized codes (see Martello and Toth[17]). Moreover, all variables z_e with $x^*_e = 0$ can be fixed to 0, because $z_e = 1$ would imply $\sigma^* \geqslant 1$. Analogously, one can set $z_e = 1$ whenever $x^*_e = 1$, because, in this case, its weight in (21) vanishes.

If $\sigma^* \geqslant 1$, then no violated cover inequality (14) exists. Otherwise, $T := \{e \in E: z_e = 1\}$ gives a most-violated such cut. In both cases, it is worth checking the extended cover inequalities (15) associated with T for violation. Notice that, because of the fact that some weights in (21) can be zero, the edgeset T, which gives the optimum in (21), is not guaranteed to be minimal with respect to property (22). Therefore, to have a stronger inequality, we make T minimal (in a greedy way) before checking the extended inequality (15) for violation.

We now outline a heuristic separation algorithm for the cycle cover inequalities (16) associated with an infeasible cycle T. The heuristic is intended to produce several candidate cycles T with large value of $x^*(T)$. To this end, we interpret the values x^*_e as weights associated with the edges, and compute a maximum-weight spanning tree on G. We then consider, in turn, the edges $e \in E^*$ not in the tree: if the addition of e to the tree induces a cycle T passing through node 1 and such that $\Sigma_{e \in T} t_e > t_0$, then we have a valid inequality (16) that we check for violation.

Conditional Cuts

We have implemented two heuristic separation procedures for conditional cuts. Let LB be the current best solution value available.

The first procedure is based on condition (20), and is embedded within the max-flow separation algorithm for GSECs described earlier. For each set S'_v, $1 \in S'_v$, therein detected, which satisfies $\Sigma_{v \in S'_v} p_v \leqslant \text{LB}$, we set $T = E(S'_v)$ and check (19) for violation.

Our second procedure is based on the observation that (19) can always be used as a conditional cut, provided that the lower bound value LB is updated by taking into account all the feasible OP solutions entirely contained in T. This amounts to computing

$$\text{LB} := \max\{\text{LB}, v(\text{OP}_T)\},$$

where $v(\text{OP}_T)$ is the optimal OP value when $x_e = 0$ is imposed for all $e \in E \backslash T$. Although the computation of $v(\text{OP}_T)$ requires exponential time in the worst case, for a sufficiently sparse edge set T, it is likely that even a simple complete enumeration scheme can succeed in determining $v(\text{OP}_T)$ within short computing time. In our implementation, we define $T := E^*$, hence ensuring that the corresponding conditional cut (19) is violated because $x^*(T) = x^*(E)$ and $y^*(V(T)) = y^*(V)$, where $x^*(E) = y^*(V)$ because of the degree equations (3). We then apply a simple algorithm for solving OP on the support graph G^*, which is based on complete enumeration. If the enumeration ends within a fixed time limit, TL, then, after the updating of LB, (19) is guaranteed to be a valid conditional cut to be added to the current LP.

4. Heuristic Algorithms

The performance of the overall branch-and-cut algorithm improves if one is capable of early detecting "good" feasible OP solutions. To this end, we propose the following heuristic procedure, working in two stages. In the first stage, a feasible cycle C is detected, which is likely to contain a large number of edges belonging to an optimal solution. In the second stage, refining procedures are applied to derive from C a (good) feasible circuit. The method is along the same lines as the heuristic proposed by Ramesh and Brown,[21] but uses LP information to guide the search. A brief outline follows.

On input of the first stage, the heuristic receives an estimate w_e, $0 \leq w_e \leq 1$, of the probability of having edge e in an optimal solution. The computation of values w_e is described in Section 5.2. We sort the edges in decreasing order of w_e, with ties broken so as to rank edges with smaller time t_e first. Then, we heuristically detect, in a greedy way, an edge subset T containing a family of node-disjoint paths with large probability of being part of an optimal solution. To be specific, we initialize $T := \emptyset$ and then consider, in turn, each edge e according to the given order: If $T \cup \{e\}$ contains a node with degree larger than 2, we reject the edge; otherwise we update $T := T \cup \{e\}$ if $T \cup \{e\}$ is cycle-free, else we stop the algorithm.

Starting with T, we obtain the required feasible cycle C by means of the following steps. We first remove from the graph all the nodes in $V\backslash\{1\}$ that are not covered by T. Then, we link the paths of T into a simple cycle, C, passing through node 1. To this end, we apply a simple nearest-neighbor scheme which starts from node 1, and iteratively moves to the nearest uncovered extreme node of a path in T. At the end of this phase, we check whether $\Sigma_{e\in C} t_e \leq t_0$. If this is not the case, we apply the following procedure to make C feasible. For any given node v covered by C, let i_v and j_v denote the two neighbors of v in C. The procedure iteratively removes a node v from C, i.e., replaces $[i_v, v]$ and $[j_v, v]$ with the shortcut $[i_v, j_v]$. At each iteration, we choose (if possible) a minimum-prize node v whose removal makes C feasible, or else a node v that minimizes the score $p_v/(t_{i_v v} + t_{j_v v} - t_{i_v j_v})$.

In the second stage of our heuristic, we receive on input the feasible cycle C, computed in the first stage, and iteratively try to improve it. At each iteration, we first perform 2-optimality edge exchanges inside C to hopefully reduce its total time. Then we try to add to C a maximum-prize node belonging to the set $Q(C)$ containing the nodes v not covered by C, and such that $\min_{[i,j]\in C}\{t_{iv} + t_{jv} - t_{ij}\} \leq t_0 - \Sigma_{e\in C} t_e$. If $Q(C) \neq \emptyset$, we perform the node insertion and repeat. Otherwise, we reapply the whole procedure on the cycle obtained from C by removing, in turn, one of its nodes.

5. A Branch-and-Cut Algorithm

We next outline the main ingredients of our branch-and-cut algorithm for the optimal OP solution. We assume the reader has some familiarity with the branch-and-cut approach as described, e.g., in Padberg and Rinaldi[20] and Jünger, Reinelt, and Rinaldi.[14]

5.1 Initialization

At the root node of the branch-decision tree, we compute a lower bound on the optimal OP value through the heuristic algorithm of Section 4, with edge weights $w_e = 0$ for all $e \in E$. In addition, we set up the first Linear Program (LP) to be solved by taking:

1. all variables y_v, $v \in V$;
2. the variables x_e associated with edges belonging to the initial heuristic solution;
3. for all $v \in V$, the variables x_e associated with the 5 smallest-cost edges $e \in \delta(v)$;
4. the total time constraint (2);
5. the n degree equations (3);
6. the lower and upper bounds on the variables.

Finally, we initialize, at empty, a data structure called the constraint pool, that contains OP constraints that are not included in the current LP.

5.2 The Cutting Plane Phase

At each node of the branch-decision tree, we determine the optimal primal and dual solutions of the current LP, say (x^*, y^*) and u^*, respectively—in case the current LP reveals infeasible, we introduce artificial variables with very large negative prize. Notice that the value of the primal solution, namely $\Sigma_{v\in V} p_v y^*_v$, is not guaranteed to give an upper bound on the optimal OP value, because the current LP contains only a subset of the x-variables. We then enter the so-called pricing phase, in which we use the dual solution u^* to compute the reduced cost $\bar{c}_e$ of the variables x_e that are not part of the current LP, which are, by default, set to 0. If some variables price-out with the wrong sign (i.e., $\bar{c}_e > 0$), then we add them to the LP, and reoptimize with the primal simplex algorithm. To keep the size of the LP as small as possible, we never add more than 100 variables at each round of pricing (chosen among those with largest reduced costs). We iterate the pricing loop until all variables price-out correctly, i.e., until the current LP value, say UB, is guaranteed to be an upper bound on the optimal OP value. In this case, if the current node is not fathomed we enter a purging phase. Let LB denote the value of the best OP solution known so far. We remove from the current LP:

1. all variables x_e with $\lfloor \text{UB} + 4\bar{c}_e \rfloor \leq \text{LB}$
2. all the constraints that have been slack in the last 5 iterations, or whose slack exceeds 0.01.

Moreover, at the root branch-decision node, we fix to 0 all the variables x_e with $\lfloor \text{UB} + \bar{c}_e \rfloor \leq \text{LB}$, and to 1 all the variables with $\lfloor \text{UB} - \bar{c}_e \rfloor \leq \text{LB}$ (this latter condition may only apply to LP variables at their upper bound).

We next enter the separation phase, in which constraints violated by (x^*, y^*) are identified and added to the current LP. We apply the separation algorithms described in Section 3. We first search the constraint pool. Then we check in sequence: the logical constraints (12), the GSECs (4), the 2-matching constraints (13), the cover inequalities (15)–(16), the path inequalities (17), and the conditional cuts (19). As to the time limit TL for the enumeration required in the con-

ditional cut separation, we set TL := 5 · TS, where TS is the computing time so far spent in the last round of separation. Whenever a separation procedure succeeds in finding violated cuts, we exit the separation sequence and add all the cuts found to the LP.

To reduce tailing off phenomena, we resort to branching whenever the upper bound did not improve by at least 0.001 in the last 10 cutting-plane iterations of the current branching node.

At every fifth application of the separation algorithm, we try to improve the current best OP solution through the heuristic algorithm described in Section 4. As to the values w_e required by the algorithm, we set $w_e := x^*_e$ for all $e \in E$. This choice computationally proved very effective and typically produces very tight approximate solutions. An additional heuristic is embedded within our second separation procedure for conditional cuts (19), as described in Section 3. Indeed, the enumeration of the OP solutions contained in the support graph of x^*, therein required, can, in some cases, improve the current LB.

5.3 The Overall Algorithm

At the root node of the branch-decision tree, we initialize the current LP and heuristic solution (as described in Section 5.1) and enter the cutting-plane phase outlined in Section 5.2. When all separation algorithms fail, if the current node is not fathomed, we would resort to branching. However, we have implemented (for the root node only) the following alternative scheme.

According to our computational experience, the conditional cut associated with the support graph $G^* = (V(E^*), E^*)$ of the current LP solution (x^*, y^*), namely

$$x(E^*) \leq y(V(E^*)) - 1, \qquad (24)$$

is quite effective in closing the integrality gap. Unfortunately, for rather dense G^* our simple enumeration scheme is unlikely to complete the enumeration of all possible OP solutions contained in G^* within the short time limit allowed. We decided to add the cut (24) to the LP even when this enumeration fails, and call it a branch cover cut. This choice may, however, cut off the optimal OP solution as well, if this solution is contained in G^*. We take care of this possibility by storing the graph G^*, with the aim of dealing with it at a later time. With the branch cover cut added to the LP, we then re-enter the cutting plane phase, until, again, all separations fail. Then, if needed, the whole scheme is iterated: we add a new branch cover cut, store the current support graph G^*, and reenter the cutting plane phase.

In this way, we produce and store a sequence of support graphs, say $G^*_i = (V(E^*_i), E^*_i)$ for $i = 1, \ldots, k$, until the root node is fathomed. At this point, the computation is not over because we have to consider the best OP solution within each graph $G^*_1, \ldots, G^*_k$ or, alternatively, within the "union" of these graphs, defined as $\tilde{G} = (V(\tilde{E}), \tilde{E} := \cup_{i=1}^{k} E^*_i)$. To this end, we remove from the constraint pool all the branch cover cuts, and re-apply our branch-and-cut algorithm on the OP instance associated with $\tilde{G}$. To guarantee the convergence of the overall algorithm, we inhibit the generation of branch cover cuts in this second branch-and-cut round. Instead, whenever a branch-decision node cannot be fathomed we branch, in a traditional way, by fixing $x_f = 0$ or $x_f = 1$ for a variable x_f chosen as follows. We select the 15 fractional variables x_e with x^*_e closest to 0.5. For each such candidate branching variable x_e, we compute two values, say UB^0_e and UB^1_e, by solving the current LP amended by with the additional constraint $x_e = 0$ and $x_e = 1$, respectively. Then, we choose the actual branching variable x_f as the one that maximizes the score $0.75 \cdot UB^0_e + 0.25 \cdot UB^1_e$.

As explained, our branch-and-cut scheme works in two stages. In the first stage, we avoid branching by adding branch cover cuts. In the second stage, we work on a sparse graph $\tilde{G}$ (resulting from the branch cover cuts produced in the first stage), and use a classical branching strategy to close the integrality gap. Our computational experience shows that the overall scheme typically performs better than (although does not dominate) the classical one. Indeed, the second stage takes advantage from a large number of relevant cuts (produced in the first stage and stored in the constraint pool), as well as a very tight approximate OP solution. On the other hand, for some instances, the first stage exhibits a slow convergence in the last iterations, because of tailing-off phenomena. To contrast this behavior, we allow for variable branching even in the first stage. Namely, in each node, we resort to branching after the addition of five branch cover cuts.

6. Computational Results

The branch-and-cut algorithm described in the previous sections was implemented in C language, and run on a DEC station 5000/240 and on an Hewlett Packard Apollo 9000/720 computer. CPLEX 3.0 was used as LP solver. We considered three different classes of test problems.

The first problem class (Class I) includes 15 instances from the OP and Vehicle Routing Problem (VRP) literature. Problems OP21, OP32, and OP33 are OP instances introduced by Tsiligirides,[25] with travel times multiplied by 100 and then rounded to the nearest integer. Problems ATT48, EIL30, EIL31, EIL33, EIL51, EIL76, EIL101, and GIL262 are VRP instances taken from the Traveling Salesman Problem Library TSPLIB 2.1 of Reinelt.[23] Problems CMT101, CMT121, CMT151, and CMT200 are VRP instances from Christofides, Mingozzi, and Toth.[7] For all the VRP instances, the customer demands are interpreted as node prizes.

The second problem class (Class II) includes all the TSP instances contained in library TSPLIB 2.1 involving up to 400 nodes (problems ATT48 to RD400). For these instances, the node prizes p_j for $j \in V\backslash\{1\}$ have been generated in three different ways:

- Generation 1: $p_j := 1$;
- Generation 2: $p_j := 1 + (7141 \cdot j + 73) \bmod (100)$;
- Generation 3: $p_j := 1 + \lfloor 99 \cdot t_{1j}/\theta \rfloor$, where $\theta := \max_{i \in V\backslash\{1\}} t_{1i}$.

Generation 1 produces OP instances in which the goal is to cover as many nodes as possible, as occurs in some applications. Generation 2 is intended to produce pseudorandom prizes in range [1, 100], whereas Generation 3 leads to more

Table I. Results for Problems of Class I (OP and VRP instances)

Name	t_0	r-time	%-LB	%-UB	nodes	cuts	opt-val	%-visited	time
$\alpha = 0.25$									
op21	1150	0.6	0.0	0.0	1	64	90	33.3	0.6
op32	2064	1.2	0.0	0.0	1	80	70	25.0	1.2
op33	2439	0.8	0.0	0.0	1	72	250	36.4	0.8
att48	2657	2.6	0.0	0.0	1	129	17	37.5	2.6
eil30	96	2.8	0.0	0.0	1	140	2650	20.0	2.8
eil31	52	0.3	0.0	0.0	1	21	535	38.7	0.3
eil33	111	1.1	0.0	0.0	1	63	800	9.1	1.1
eil51	107	2.2	0.0	0.0	1	141	264	27.5	2.2
eil76	135	48.7	0.0	0.0	1	542	490	30.3	48.7
eil101	158	177.3	0.0	0.5	3	900	572	31.7	189.7
cmt101	127	57.8	0.0	0.0	1	586	530	28.7	57.8
cmt121	137	347.9	1.5	1.0	9	1315	412	33.9	612.5
cmt151	175	128.4	0.0	0.0	1	1066	824	29.1	128.4
cmt200	191	130.7	0.1	0.0	2	1161	1205	33.0	299.1
gil262	595	2541.6	1.4	1.9	18	4800	4466	29.6	t.l.
$\alpha = 0.50$									
op21	2299	0.7	0.0	0.0	1	58	205	61.9	0.7
op32	4127	1.3	0.0	0.0	1	91	160	56.2	1.3
op33	4878	1.8	0.0	0.0	1	109	500	63.6	1.8
att48	5314	0.8	0.0	0.0	1	55	30	64.6	0.8
eil30	191	5.2	0.0	0.0	1	170	7600	26.7	5.2
eil31	103	0.5	0.0	0.0	1	32	747	58.1	0.5
eil33	221	8.5	0.0	0.0	1	215	16220	48.5	8.5
eil51	213	2.4	0.0	0.0	1	150	508	54.9	2.4
eil76	269	3.1	0.0	0.0	1	118	907	59.2	3.1
eil101	315	5.6	0.0	0.0	1	159	1049	57.4	5.6
cmt101	253	36.9	1.0	0.0	2	514	1030	56.4	55.2
cmt121	273	411.1	0.0	0.8	39	1350	715	52.9	1525.6
cmt151	350	131.8	0.1	0.1	5	451	1537	55.6	167.3
cmt200	382	147.4	0.0	0.0	17	859	2198	62.0	596.3
gil262	1189	365.8	0.2	0.2	35	2370	8456	54.8	3252.7
$\alpha = 0.75$									
op21	3449	1.7	0.0	0.0	1	82	315	71.4	1.7
op32	6191	0.8	0.0	0.0	1	73	230	71.9	0.8
op33	7317	1.1	0.0	0.0	1	70	660	84.8	1.1
att48	7971	1.1	0.0	0.0	1	66	39	83.3	1.1
eil30	286	0.9	0.0	0.0	1	63	11550	73.3	0.9
eil31	155	1.6	0.0	0.0	1	32	865	87.1	1.6
eil33	331	1.7	0.0	0.0	1	85	26380	81.8	1.7
eil51	320	60.0	0.0	0.3	5	143	690	76.5	77.6
eil76	404	56.6	0.2	0.4	71	479	1186	84.2	457.6
eil101	472	4.7	0.0	0.0	1	98	1336	81.2	4.7
cmt101	379	130.7	0.0	0.0	1	676	1480	75.2	130.7
cmt121	409	50.9	0.0	0.0	1	495	1134	76.9	50.9
cmt151	525	114.6	0.2	0.0	7	368	2003	79.5	380.3
cmt200	573	127.9	8.2	0.2	337	2668	2881	84.0	17389.9
gil262	1784	436.5	8.1	0.2	253	2379	11195	77.2	17512.0

difficult instances in which large prizes are assigned to the nodes far away from the depot.

For the third problem class (Class III), we have obtained the random instances by using the original Laporte and Martello[15] code. In this class, both prizes and travel times are generated as uniformly random integers in range [1, 100], with travel times triangularized through shortest-path computation.

Table II. Additional Results for Class I Problems

Name	t-LP	t-sep	cuts	log	gsec	2-mat	cover	path	cond	b-cov
$\alpha = 0.25$										
op21	0.4	0.1	64	21	31	0	0	0	11	0
op32	0.8	0.1	80	32	37	1	0	0	9	0
op33	0.4	0.2	72	26	27	3	0	0	15	0
att48	1.5	0.5	129	44	65	8	0	0	11	0
eil30	2.2	0.2	140	54	68	0	0	0	17	0
eil31	0.2	0.0	21	11	3	1	0	0	5	0
eil33	0.8	0.0	63	32	29	0	0	0	1	0
eil51	1.4	0.3	141	48	77	1	0	0	14	0
eil76	27.7	15.4	542	82	271	5	16	11	156	0
eil101	88.3	80.8	900	129	539	41	5	25	156	4
cmt101	41.3	8.1	586	157	354	6	2	0	66	0
cmt121	271.4	239.2	1315	223	687	59	22	249	64	10
cmt151	79.4	29.7	1066	172	739	33	5	0	116	0
cmt200	148.7	101.7	1161	211	634	42	8	0	263	2
gil262	9573.5	1581.3	4800	476	3422	132	10	205	488	66
$\alpha = 0.50$										
op21	0.4	0.1	58	27	14	0	0	0	16	0
op32	0.9	0.2	91	31	46	2	1	0	10	0
op33	1.0	0.5	109	31	47	2	0	0	28	0
att48	0.4	0.2	55	23	17	11	0	0	3	0
eil30	3.6	0.8	170	43	84	0	4	0	38	0
eil31	0.2	0.2	32	15	5	1	1	0	9	0
eil33	5.4	2.0	215	42	86	2	14	20	50	0
eil51	1.6	0.4	150	48	54	5	0	0	42	0
eil76	1.4	1.1	118	50	49	7	0	0	11	0
eil101	2.7	1.5	159	61	59	19	1	0	18	0
cmt101	23.9	16.0	514	114	362	14	8	6	8	1
cmt121	503.5	652.9	1350	189	719	86	82	63	172	38
cmt151	27.8	121.0	451	127	210	61	7	0	39	6
cmt200	84.8	422.6	859	177	449	82	38	0	94	18
gil262	740.0	1873.6	2370	262	1131	154	83	49	644	46
$\alpha = 0.75$										
op21	0.8	0.6	82	22	17	2	2	0	38	0
op32	0.5	0.1	73	29	16	0	0	0	27	0
op33	0.5	0.4	70	15	16	4	0	0	34	0
att48	0.5	0.4	66	24	26	6	0	0	9	0
eil30	0.6	0.1	63	14	19	2	0	0	27	0
eil31	0.5	0.9	32	14	0	0	5	0	12	0
eil33	0.8	0.6	85	20	16	4	1	0	43	0
eil51	4.4	69.9	143	20	24	29	16	0	48	5
eil76	50.7	328.2	479	59	111	99	18	0	138	53
eil101	1.7	2.1	98	26	30	9	1	0	31	0
cmt101	36.2	86.5	676	88	355	47	14	0	169	2
cmt121	24.9	19.2	495	88	277	63	2	0	64	0
cmt151	26.8	332.5	368	69	125	39	25	0	97	12
cmt200	1934.7	13054.5	2668	120	560	1129	17	0	515	326
gil262	2020.3	12874.4	2379	165	638	789	13	1	547	225

For all problem classes, we define the maximum total travel time as $t_0 := \lceil \alpha \cdot \nu(\text{TSP}) \rceil$, where $\nu(\text{TSP})$ is the length of the shortest Hamiltonian tour, and α is a given parameter. For all instances taken from TSPLIB, the value $\nu(\text{TSP})$ is provided within the library. For problems OP21, OP32, OP33, CMT101, CMT121, CMT151, and CMT200, respectively, we used the following values for $\nu(\text{TSP})$: 4598, 8254, 9755, 505, 545, 699, and 764. For the random problems of

Table III. Results for Class II (TSPLIB Instances) and Generation 1

Name	t_0	r-time	%-LB	%-UB	nodes	cuts	opt-val	%-visited	time
spain47	3101	0.7	0.0	0.0	1	74	28	59.6	0.7
europ47	13353	21.6	0.0	3.1	3	114	32	68.1	22.7
att48	5314	0.7	0.0	0.0	1	61	31	64.6	0.7
gr48	2523	1.1	0.0	0.0	1	96	31	64.6	1.1
hk48	5731	1.7	0.0	0.0	1	105	30	62.5	1.7
eil51	213	1.2	0.0	0.0	1	114	29	56.9	1.2
brazil58	12698	3.2	0.0	0.0	1	120	46	79.3	3.2
st70	338	5.3	0.0	0.0	1	193	43	61.4	5.3
eil76	269	5.7	0.0	0.0	1	216	47	61.8	5.7
pr76	54080	50.9	0.0	0.0	1	322	49	64.5	50.9
gr96	27605	113.8	0.0	1.6	3	518	64	66.7	121.1
rat99	606	53.5	0.0	0.0	1	517	52	52.5	53.5
kroa100	10641	27.1	0.0	0.0	1	440	56	56.0	27.1
krob100	11071	156.7	1.7	0.0	3	616	58	58.0	326.9
kroc100	10375	50.7	0.0	0.0	1	518	56	56.0	50.7
krod100	10647	32.0	0.0	0.0	1	434	59	59.0	32.0
kroe100	11034	82.9	0.0	1.8	67	863	57	57.0	776.0
rd100	3955	30.2	0.0	0.0	1	425	61	61.0	30.2
eil101	315	7.1	0.0	0.0	1	216	64	63.4	7.1
lin105	7190	78.8	0.0	1.5	3	540	66	62.9	83.4
pr107	22152	86.3	0.0	0.0	1	630	54	50.5	86.3
gr120	3471	17.5	0.0	0.0	1	320	75	62.5	17.5
pr124	29515	41.2	0.0	0.0	1	480	75	60.5	41.2
bier127	59141	73.7	0.0	0.0	1	379	103	81.1	73.7
pr136	48386	214.2	0.0	0.0	1	990	71	52.2	214.2
gr137	34927	178.6	0.0	0.0	1	553	81	59.1	178.6
pr144	29269	240.3	0.0	0.0	1	1170	77	53.5	240.3
kroa150	13262	582.2	0.0	1.2	85	2594	86	57.3	4669.0
krob150	13065	145.6	0.0	0.0	1	729	87	58.0	145.6
pr152	36841	204.6	0.0	0.0	1	917	77	50.7	204.6
u159	21040	497.6	0.0	0.0	1	1912	93	58.5	497.6
rat195	1162	331.9	0.0	0.0	1	1323	102	52.3	331.9
d198	7890	716.3	0.0	0.0	1	1431	123	62.1	716.3
kroa200	14684	395.0	0.0	0.0	1	1254	117	58.5	395.0
krob200	14719	683.6	0.0	0.0	1	1635	119	59.5	683.6
gr202	20080	150.6	0.0	0.0	1	603	147	72.8	150.6
ts225	63322	9.7	0.0	0.8	107	6040	125	55.5	t.l.
pr226	40185	1955.3	0.0	5.2	25	1019	134	59.3	t.l.
gr229	1765	75.0	0.0	0.0	1	431	176	76.9	75.0
gil262	1189	120.6	0.0	0.0	1	789	158	60.3	120.6
pr264	24568	2860.2	0.0	0.0	1	1694	132	50.0	2860.2
pr299	24096	5726.3	1.2	1.2	8	4524	162	54.2	14244.0
lin318	21045	2558.0	0.0	0.5	5	2653	205	64.5	3169.9
rd400	7641	874.0	1.7	0.4	29	1821	239	59.8	4272.5

Class III, we used the approximate value computed by the original Laporte–Martello code, namely ν(TSP) := $\lfloor 0.95 \cdot$ UB(TSP) + 0.5$\rfloor$, where UB(TSP) is the length of the tour obtained by the heuristic algorithm proposed by Rosenkrantz, Stearns, and Lewis.[24]

Tables I through VI report on the computational behavior of our branch-and-cut code on the various classes of instances. Each table (except Table II) gives:

Name: The problem name;

t_o: the maximum total time (only for Classes I and II);

r-time: the total time spent at the root node;

%-LB: the percentage ratio (optimum − LB)/optimum, where LB is the value of the best heuristic solution computed at the root node;

Table IV. Results for Class II (TSPLIB Instances) and Generation 2

Name	t_0	r-time	%-LB	%-UB	nodes	cuts	opt-val	%-visited	time
spain47	3101	12.3	0.0	0.0	1	202	1645	53.2	12.3
europ47	13353	48.3	0.0	0.0	1	125	1865	61.7	48.3
att48	5314	3.9	0.0	0.0	1	133	1717	64.6	3.9
gr48	2523	18.0	0.0	0.0	1	196	1761	56.2	18.0
hk48	5731	7.1	0.0	0.0	1	183	1614	56.2	7.1
eil51	213	30.7	0.0	0.0	1	185	1674	52.9	30.7
brazil58	12698	7.8	0.0	0.0	1	230	2220	72.4	7.8
st70	338	147.2	0.0	0.4	7	593	2286	55.7	181.0
eil76	269	7.2	0.0	0.0	1	208	2550	53.9	7.2
pr76	54080	58.6	0.0	0.2	3	271	2708	57.9	62.0
gr96	27605	399.3	0.0	0.1	5	1185	3425	63.5	453.1
rat99	606	125.4	0.0	0.0	1	897	2944	46.5	125.4
kroa100	10641	67.8	0.0	0.0	1	558	3212	55.0	67.8
krob100	11071	259.0	0.0	0.3	7	1414	3241	49.0	481.4
kroc100	10375	207.6	0.1	0.4	7	815	2947	48.0	316.2
krod100	10647	237.2	0.0	0.3	9	757	3307	54.0	334.2
kroe100	11034	285.5	1.4	1.9	43	1518	3090	54.0	1433.9
rd100	3955	27.8	0.0	0.0	1	387	3359	57.0	27.8
eil101	315	51.5	0.0	0.4	17	711	3655	57.4	296.5
lin105	7190	151.9	0.0	0.1	3	556	3544	58.1	163.6
pr107	22152	99.4	0.0	0.0	1	678	2667	50.5	99.4
gr120	3471	303.4	0.0	0.2	21	1022	4371	57.5	650.0
pr124	29515	79.4	0.0	0.0	1	796	3917	59.7	79.4
bier127	59141	73.2	0.0	0.1	7	439	5383	77.2	245.8
pr136	48386	194.3	0.0	0.0	1	1011	4309	47.8	194.3
gr137	34927	797.9	0.0	0.3	81	1929	4294	57.7	3193.0
pr144	29269	668.0	0.0	0.7	11	1579	4003	51.4	1409.0
kroa150	13262	460.1	0.6	0.9	47	2876	4918	54.0	3950.6
krob150	13065	735.7	0.0	0.1	5	1795	4869	52.7	1018.1
pr152	36841	188.6	0.0	0.0	1	836	4279	48.0	188.6
u159	21040	518.0	0.4	0.2	21	1853	4960	54.1	1772.4
rat195	1162	1750.1	0.0	0.2	13	2691	5791	48.2	2498.6
d198	7890	1337.8	0.1	0.1	27	1999	6670	56.1	2517.1
kroa200	14684	515.2	0.0	0.1	15	1147	6547	54.5	805.1
krob200	14719	1240.7	0.1	0.1	17	2563	6419	50.5	3522.8
gr202	20080	441.7	0.8	0.0	31	1945	7848	65.8	3847.6
ts225	63322	763.3	0.0	0.1	5	1627	6834	54.2	1195.5
pr226	40185	3973.9	0.1	0.3	27	1842	6615	46.6	t.l.
gr229	1765	329.5	0.5	0.1	47	1415	9187	72.1	4261.4
gil262	1189	2783.0	0.1	0.2	23	2080	8321	50.8	5574.6
pr264	24568	4253.3	0.0	0.0	1	2211	6654	50.0	4253.3
pr299	24096	10803.8	0.0	0.0	5	1900	9161	49.8	t.l.
lin318	21045	1370.0	0.0	0.0	41	1392	10900	60.7	t.l.
rd400	7641	837.6	0.1	0.2	76	3721	13648	54.5	t.l.

%-UB: the percentage ratio (UB − optimum)/optimum, where UB is the upper bound computed at the root node;

nodes: the total number of nodes generated (1 means that the problem required no branching);

cuts: the total number of cuts generated (including the total time restriction (2));

opt-val: the optimal solution value (only for classes I and II);

%-visited: the percentage number of nodes visited by the optimal solution;

time: the total computing time spent by the branch-and-cut code.

Table V. Results for Class II (TSPLIB Instances) and Generation 3

Name	t_0	r-time	%-LB	%-UB	nodes	cuts	opt-val	% visited	time
spain47	3101	16.9	0.0	0.0	1	207	1443	55.3	16.9
europ47	13353	34.6	0.0	0.0	1	244	1282	59.6	34.6
att48	5314	180.2	0.0	0.8	7	532	1049	60.4	251.8
gr48	2523	27.5	0.0	0.0	1	316	1480	64.6	27.5
hk48	5731	3.5	0.0	0.0	1	101	1764	58.3	3.5
eil51	213	19.5	0.0	0.0	1	214	1399	52.9	19.5
brazil58	12698	2.8	0.0	0.0	1	112	1702	70.7	2.8
st70	338	70.6	0.0	0.0	1	623	2108	51.4	70.6
eil76	269	49.6	0.0	0.0	1	383	2467	57.9	49.6
pr76	54080	34.0	0.0	0.0	1	286	2430	61.8	34.0
gr96	27605	189.1	0.1	0.8	9	804	3182	65.6	416.6
rat99	606	481.1	0.0	0.1	3	993	2908	45.5	487.4
kroa100	10641	144.8	0.0	0.4	13	536	3211	51.0	248.8
krob100	11071	134.1	0.0	0.1	3	564	2804	49.0	138.9
kroc100	10375	228.1	0.0	0.0	1	938	3155	52.0	228.1
krod100	10647	167.8	0.0	0.3	9	559	3167	56.0	230.3
kroe100	11034	184.4	0.0	0.0	1	941	3049	45.0	184.4
rd100	3955	644.9	0.0	0.3	11	1423	2926	55.0	1032.3
eil101	315	120.0	0.0	0.1	7	474	3345	59.4	186.7
lin105	7190	325.9	1.4	1.1	19	1407	2986	55.2	1121.1
pr107	22152	632.4	0.1	3.2	1885	9926	1877	26.2	17609.0
gr120	3471	145.5	0.0	0.0	1	811	3779	57.5	145.5
pr124	29515	1377.8	6.2	5.6	77	4540	3557	57.3	11487.2
bier127	59141	139.3	0.6	0.5	73	1022	2365	68.5	2001.2
pr136	48386	861.6	0.0	0.2	5	1146	4390	51.5	958.5
gr137	34927	2401.9	45.5	0.1	5	5386	3979	51.8	4958.7
pr144	29269	1573.8	0.0	0.1	65	2632	3809	43.1	t.l.
kroa150	13262	269.7	0.1	0.6	181	1646	5039	52.7	3828.9
krob150	13065	1112.5	0.0	0.1	13	1472	5314	56.7	1363.9
pr152	36841	1099.0	0.7	1.3	391	3159	3905	48.7	13736.7
u159	21040	1308.4	0.0	0.2	5	2171	5272	52.8	1447.2
rat195	1162	3672.2	0.1	0.1	9	1528	6195	47.7	3975.4
d198	7890	1810.0	0.0	0.2	197	2361	6320	61.6	8635.7
kroa200	14684	3116.5	0.0	0.2	41	2161	6123	51.5	6548.9
krob200	14719	642.6	0.0	0.1	9	905	6266	51.0	783.7
gr202	20080	654.2	0.5	0.3	298	1947	8632	71.3	11113.5
ts225	63322	3437.6	0.0	0.4	27	1598	7575	55.1	5821.8
pr226	40185	3379.5	16.0	0.1	51	5310	6993	52.2	7923.2
gr229	1765	1667.1	0.0	0.0	11	1613	6347	67.7	1891.5
gil262	1189	6177.4	0.2	0.0	27	2386	9246	56.5	9574.0
pr264	24568	4011.3	0.0	0.0	1	2625	8137	39.8	4011.3
pr299	24096	14699.4	0.0	0.0	2	1787	10358	49.8	t.l.
lin318	21045	8597.5	0.0	0.0	12	1308	10382	60.7	t.l.
rd400	7641	14257.1	0.0	0.0	3	1418	13229	55.8	t.l.

The computing times reported are expressed in seconds, and refer to CPU time on an HP Apollo 9000/720 computer running at 80 MHz, with the following figures: 59 SPECs, 58 MIPs, and 18 MFlops. We imposed a time limit of 18,000 seconds (5 hours) for each run. For the instances exceeding the time limit, we report "t.l." in the time column, and compute the corresponding results by considering the best available as the optimal solution. Hence, for the time-limit instances the column %-UB gives an upper bound on the percentage approximation error.

Table I refers to the instances of Class I. We considered 3 values for the parameter α, namely $\alpha = 0.25$, 0.50, and 0.75. For this problem class, we also report, in Table II, additional information on the overall time spent within the LP solver (t-LP) and the separation procedures (t-sep), and on the number of logical (log), GSEC (gsec), 2-matching (2-mat),

Table VI. Average (maximum) Results over 10 Random Instances of Class III

n	α	r-time	%-LB	%-UB	nodes	cuts	%-visited	time
25	0.2	1.1 (3.4)	0.0 (0.0)	0.0 (0.0)	1.1 (2.0)	63.4 (92.0)	28.8 (40.0)	1.2 (3.4)
25	0.4	38.8 (290.4)	0.0 (0.0)	0.9 (5.9)	3.8 (15.0)	138.2 (301.0)	56.8 (64.0)	42.7 (296.6)
25	0.6	46.6 (310.2)	0.0 (0.0)	0.3 (2.3)	2.6 (7.0)	101.0 (272.0)	74.0 (80.0)	63.4 (447.8)
25	0.8	42.1 (160.0)	0.0 (0.0)	0.4 (1.5)	4.0 (17.0)	91.5 (184.0)	86.0 (92.0)	51.5 (191.6)
50	0.2	41.5 (374.9)	0.0 (0.0)	0.4 (3.2)	1.6 (5.0)	169.6 (365.0)	32.8 (40.0)	53.8 (484.4)
50	0.4	32.1 (169.4)	0.0 (0.2)	0.5 (1.1)	10.0 (27.0)	220.2 (354.0)	59.8 (66.0)	99.6 (377.8)
50	0.6	30.8 (78.4)	0.3 (1.1)	0.3 (0.5)	21.8 (88.0)	263.1 (829.0)	76.4 (80.0)	147.0 (461.3)
50	0.8	10.5 (25.5)	0.2 (1.1)	0.1 (0.6)	10.4 (67.0)	128.1 (421.0)	90.2 (94.0)	58.9 (279.6)
100	0.2	61.3 (201.5)	0.0 (0.3)	0.3 (1.1)	8.2 (31.0)	359.6 (686.0)	35.3 (39.0)	149.6 (706.2)
100	0.4	35.9 (66.4)	0.2 (0.7)	0.2 (0.4)	28.4 (81.0)	403.5 (802.0)	60.9 (66.0)	340.3 (785.0)
100	0.6	33.7 (63.5)	0.4 (1.2)	0.1 (0.2)	34.6 (77.0)	421.4 (552.0)	80.8 (84.0)	445.2 (685.1)
100	0.8	41.9 (101.3)	0.3 (1.0)	0.0 (0.1)	35.4 (237.0)	273.9 (940.0)	93.2 (95.0)	403.8 (1825.9)
150	0.2	63.5 (212.7)	0.0 (0.0)	0.1 (0.8)	3.8 (9.0)	353.5 (684.0)	33.7 (37.3)	112.6 (343.9)
150	0.4	64.6 (145.8)	0.4 (1.3)	0.1 (0.2)	26.3 (74.0)	448.4 (1336.0)	59.5 (64.0)	493.1 (1460.5)
150	0.6	54.7 (95.9)	0.2 (0.6)	0.1 (0.2)	25.3 (89.0)	335.7 (765.0)	79.2 (80.7)	525.0 (1491.1)
150	0.8	67.8 (123.8)	0.3 (0.5)	0.0 (0.1)	21.9 (127.0)	258.3 (802.0)	95.5 (97.3)	549.9 (2491.6)
200	0.2	85.9 (175.9)	0.0 (0.2)	0.1 (0.3)	9.0 (27.0)	394.9 (642.0)	32.9 (36.5)	175.4 (473.0)
200	0.4	95.8 (200.6)	0.2 (0.9)	0.1 (0.1)	39.6 (155.0)	560.2 (1229.0)	59.2 (63.0)	986.2 (3138.5)
200	0.6	115.7 (235.1)	0.5 (2.9)	0.0 (0.1)	71.0 (226.0)	612.2 (1216.0)	80.4 (83.5)	2062.2 (5142.3)
200	0.8	81.0 (158.9)	0.2 (0.6)	0.0 (0.0)	7.3 (13.0)	186.3 (341.0)	97.3 (100.0)	636.8 (1100.2)
250	0.2	139.9 (338.5)	0.0 (0.1)	0.1 (0.3)	14.4 (51.0)	460.9 (737.0)	31.3 (32.8)	308.8 (761.9)
250	0.4	113.3 (236.4)	0.2 (0.7)	0.1 (0.1)	52.7 (141.0)	583.1 (1133.0)	57.8 (60.4)	1386.4 (3504.6)
250	0.6	154.2 (485.5)	0.7 (5.9)	0.0 (0.1)	49.3 (337.0)	408.9 (1295.0)	79.7 (82.4)	1898.4 (8932.8)
250	0.8	185.2 (409.1)	1.4 (5.5)	0.0 (0.0)	22.7 (153.0)	364.4 (1924.0)	98.4 (100.0)	2850.9 (1t.l.)
300	0.2	102.7 (177.2)	0.1 (0.4)	0.1 (0.2)	15.0 (47.0)	484.8 (750.0)	30.1 (31.0)	363.5 (946.5)
300	0.4	139.1 (208.5)	0.2 (0.7)	0.0 (0.1)	43.6 (107.0)	652.4 (1202.0)	57.5 (60.7)	1816.5 (3908.2)
300	0.6	203.9 (293.2)	0.2 (0.4)	0.0 (0.0)	26.2 (57.0)	355.3 (605.0)	80.4 (82.3)	1949.5 (3895.6)
300	0.8	237.2 (846.8)	1.5 (9.2)	0.0 (0.0)	4.3 (13.0)	186.6 (628.0)	99.8 (100.0)	2038.5 (10230.2)
350	0.2	144.4 (359.4)	0.0 (0.1)	0.1 (0.1)	8.5 (23.0)	393.9 (591.0)	29.1 (30.0)	257.2 (635.1)
350	0.4	312.1 (402.0)	0.1 (0.4)	0.0 (0.0)	49.2 (129.0)	621.9 (973.0)	56.6 (58.0)	2516.6 (4837.7)
350	0.6	238.2 (477.5)	0.2 (0.4)	0.0 (0.0)	13.2 (48.0)	256.8 (568.0)	80.1 (82.0)	1493.8 (4674.1)
350	0.8	309.1 (477.4)	0.8 (5.0)	0.0 (0.0)	12.4 (48.0)	240.4 (568.0)	89.8 (100.0)	1612.1 (4669.8)
400	0.2	153.2 (299.8)	0.0 (0.1)	0.0 (0.1)	15.4 (67.0)	395.2 (664.0)	28.2 (29.0)	439.1 (1392.5)
400	0.4	181.5 (314.4)	0.2 (0.7)	0.0 (0.0)	41.8 (214.0)	469.6 (1021.0)	55.4 (57.2)	2437.3 (9675.7)
400	0.6	294.2 (550.3)	3.0 (22.0)	0.0 (0.0)	7.3 (20.0)	216.5 (510.0)	79.7 (81.8)	1844.3 (6206.2)
400	0.8	369.4 (624.0)	1.2 (6.1)	0.0 (0.0)	1.5 (4.0)	101.4 (267.0)	100.0 (100.0)	763.9 (3883.8)
450	0.2	195.5 (290.2)	0.1 (0.2)	0.0 (0.1)	32.7 (142.0)	528.7 (961.0)	27.7 (28.2)	1008.7 (3257.9)
450	0.4	265.9 (359.6)	0.6 (4.0)	0.0 (0.0)	9.3 (35.0)	289.2 (388.0)	54.7 (55.3)	837.0 (1909.9)
450	0.6	360.2 (716.8)	1.9 (9.5)	0.0 (0.1)	10.9 (35.0)	382.2 (1188.0)	79.5 (81.6)	6050.5 (3t.l.)
450	0.8	550.1 (970.8)	1.0 (2.7)	0.0 (0.0)	2.1 (6.0)	125.1 (337.0)	100.0 (100.0)	1954.4 (8008.6)
500	0.2	189.6 (726.8)	0.0 (0.0)	0.0 (0.0)	5.4 (23.0)	300.5 (493.0)	27.0 (27.2)	325.4 (1422.9)
500	0.4	317.4 (501.2)	0.3 (0.8)	0.0 (0.0)	9.9 (22.0)	322.9 (482.0)	53.6 (54.0)	1418.0 (3034.8)
500	0.6	408.4 (639.3)	1.1 (4.5)	0.0 (0.1)	9.7 (30.0)	284.2 (811.0)	78.8 (79.6)	5327.9 (1t.l.)
500	0.8	650.2 (1206.3)	1.4 (3.3)	0.0 (0.0)	2.4 (8.0)	116.2 (399.0)	100.0 (100.0)	2454.0 (11860.4)

cover (cover), path (path), conditional (cond), and branch cover (b-cov) constraints generated.

Tables III through V refer to the instances of Class II, with prizes computed according to Generation 1, 2, and 3, respectively. The parameter α has been set to 0.50. We also considered cases $\alpha = 0.25$ and $\alpha = 0.75$, with comparable results.

Table VI reports average (maximum) results over 10 random instances belonging to Class III, with $\alpha = 0.2$, 0.4, 0.6, and 0.8, and $n = 25$, 50, 100, 150, 200, 250, 300, 350, 400, 450, and 500. Larger instances could be solved as well, because, within this class, the computing time tends to increase very slowly with n for $n \geq 200$. As a comparison, the branch-and-bound algorithm of Laporte and Martello[15] ran into diffi-

Table VII. Average (maximum) Results over 5 Random Instances of Class IV

n	α	N_1	N_2	N_3	r-time	%-LB	%-UB	nodes	cuts	%-visited	time
21	0.1	5	5	5	0.2 (0.3)	0.0 (0.0)	0.0 (0.0)	1.0	20.8	14.3	0.2
21	0.3	5	5	5	0.6 (1.3)	0.0 (0.0)	0.0 (0.0)	1.0	54.6	34.3	0.6
21	0.5	5	5	5	1.0 (3.0)	0.0 (0.0)	0.0 (0.0)	1.0	73.8	56.2	1.0
21	0.7	5	5	5	1.6 (2.4)	0.0 (0.0)	0.0 (0.0)	1.0	79.2	67.6	1.6
21	0.9	5	5	5	23.9 (117.8)	0.0 (0.0)	1.3 (6.1)	1.8	46.8	90.5	24.4
41	0.1	5	5	5	1.4 (2.9)	0.0 (0.0)	0.0 (0.0)	1.0	87.6	11.2	1.4
41	0.3	5	5	5	21.4 (92.2)	0.0 (0.0)	0.0 (0.0)	1.0	210.4	36.1	21.4
41	0.5	5	5	5	56.6 (209.1)	0.0 (0.0)	0.1 (0.5)	1.8	214.4	54.1	69.7
41	0.7	5	5	5	44.3 (91.0)	0.0 (0.0)	0.6 (1.2)	19.4	264.0	74.1	129.2
41	0.9	5	5	5	23.5 (109.6)	0.0 (0.0)	0.1 (0.6)	8.6	106.4	89.3	62.7
61	0.1	5	5	5	4.9 (9.0)	0.0 (0.0)	0.0 (0.0)	1.0	168.6	11.1	4.9
61	0.3	5	5	5	74.7 (199.0)	0.0 (0.0)	0.8 (2.9)	9.4	466.4	34.4	232.8
61	0.5	5	5	5	48.0 (85.7)	0.0 (0.1)	0.3 (0.7)	3.0	293.2	56.1	74.5
61	0.7	5	5	5	52.6 (94.4)	0.2 (0.5)	0.5 (1.4)	13.4	365.4	70.8	226.7
61	0.9	5	5	5	42.8 (49.2)	0.1 (0.5)	0.3 (0.7)	18.6	300.6	88.9	190.0
81	0.1	5	5	5	30.0 (123.3)	0.0 (0.0)	0.0 (0.0)	1.0	301.4	11.6	30.0
81	0.3	5	5	5	188.4 (374.3)	0.0 (0.0)	1.3 (3.7)	11.4	878.4	35.1	435.7
81	0.5	5	5	5	54.1 (111.8)	0.0 (0.0)	0.1 (0.5)	5.4	451.8	55.8	98.7
81	0.7	5	5	4	71.6 (154.6)	0.5 (1.9)	0.4 (1.2)	39.6	611.2	74.1	443.3
81	0.9	5	5	3	66.5 (90.4)	0.4 (0.9)	0.3 (0.5)	41.8	456.4	90.6	602.5
101	0.1	5	5	5	60.5 (240.7)	0.0 (0.0)	0.8 (4.0)	3.4	559.8	11.7	193.3
101	0.3	5	5	5	186.7 (486.5)	0.0 (0.0)	0.3 (1.0)	3.0	849.6	33.5	228.7
101	0.5	5	5	5	175.3 (234.9)	0.0 (0.0)	0.3 (0.5)	11.8	806.4	55.6	350.2
101	0.7	5	5	5	70.9 (103.8)	0.3 (1.2)	0.2 (0.4)	37.8	747.6	75.6	584.5
101	0.9	5	5	1	55.2 (96.3)	1.1 (2.9)	0.3 (0.5)	222.6	872.6	90.7	2467.5
121	0.1	5	5	5	28.5 (54.2)	0.0 (0.0)	0.0 (0.0)	1.0	438.6	11.2	28.5
121	0.3	5	5	3	184.5 (406.2)	1.0 (4.9)	0.5 (1.1)	14.6	1035.8	34.2	560.8
121	0.5	5	5	4	176.5 (207.2)	0.2 (0.4)	0.5 (1.2)	37.0	1195.2	53.4	955.9
121	0.7	5	4	2	174.3 (381.0)	0.3 (1.1)	0.3 (0.6)	126.2	1293.4	72.9	2496.4
121	0.9	5	5	0	99.2 (125.9)	1.2 (3.1)	0.1 (0.4)	48.8	544.8	90.6	1065.8
141	0.1	5	5	5	108.6 (380.6)	0.0 (0.0)	0.0 (0.0)	1.0	608.0	11.2	108.6
141	0.3	5	5	3	404.7 (897.3)	0.5 (1.3)	0.5 (1.3)	21.4	1692.0	33.0	1499.4
141	0.5	5	5	1	174.2 (274.8)	0.1 (0.5)	0.2 (0.6)	15.0	988.6	53.8	641.6
141	0.7	5	2	1	200.4 (274.4)	4.1 (12.7)	0.2 (0.3)	165.8	1728.8	73.6	4750.3
141	0.9	5	5	–	175.9 (293.0)	2.0 (5.1)	0.2 (0.3)	77.8	660.8	90.4	2263.1
161	0.1	5	5	5	233.8 (502.3)	0.0 (0.0)	0.2 (0.8)	1.4	900.6	12.2	275.9
161	0.3	5	5	2	539.6 (753.2)	0.4 (0.9)	0.5 (0.9)	11.0	1638.2	34.9	1254.6
161	0.5	5	5	0	249.4 (358.6)	0.2 (0.6)	0.2 (0.4)	20.2	956.2	55.0	737.2
161	0.7	4	3	0	185.2 (242.3)	0.7 (1.4)	0.2 (0.5)	101.5	1419.5	72.8	3563.2
161	0.9	5	4	–	136.0 (221.3)	3.8 (5.7)	0.1 (0.2)	90.0	697.2	90.3	2981.2
181	0.1	5	5	5	187.7 (525.5)	0.0 (0.0)	0.0 (0.0)	1.0	821.4	12.3	187.7
181	0.3	4	3	0	758.8 (1227.7)	0.3 (1.1)	0.3 (0.7)	19.0	2540.2	35.4	2562.8
181	0.5	5	5	–	267.3 (800.5)	0.1 (0.3)	0.1 (0.3)	20.2	1320.4	54.7	1303.8
181	0.7	5	4	–	221.3 (265.2)	2.5 (12.4)	0.1 (0.2)	71.2	1258.2	73.5	3137.4
181	0.9	3	2	–	210.4 (292.4)	2.2 (6.5)	0.1 (0.1)	88.7	844.7	89.9	4277.0
201	0.1	5	5	5	397.9 (596.5)	0.0 (0.0)	0.6 (2.9)	5.8	1345.4	11.4	673.2
201	0.3	5	4	–	555.7 (814.5)	0.2 (0.4)	0.6 (1.6)	37.4	2188.2	34.3	2793.0
201	0.5	5	5	–	412.1 (572.1)	0.0 (0.0)	0.1 (0.1)	9.8	1062.6	54.5	724.1
201	0.7	4	1	–	952.7 (2821.3)	1.4 (2.6)	0.1 (0.2)	88.5	2136.2	71.8	6118.0
201	0.9	4	1	–	305.9 (461.0)	2.4 (5.6)	0.1 (0.2)	121.8	1224.5	90.4	9917.1
221	0.1	5	5	4	280.4 (417.7)	0.0 (0.0)	0.1 (0.7)	1.4	933.0	11.9	285.0
221	0.3	5	4	–	977.0 (1684.0)	0.2 (0.6)	0.4 (0.4)	28.6	2534.8	34.4	4306.0

Table VII. (*Continued*)

n	α	N_1	N_2	N_3	r-time	%-LB	%-UB	nodes	cuts	%-visited	time
221	0.5	5	3	–	460.4 (905.6)	1.2 (4.6)	0.2 (0.3)	65.4	2782.0	55.4	5135.0
221	0.7	5	2	–	451.0 (812.7)	2.1 (10.2)	0.2 (0.3)	95.0	2349.2	73.0	9242.6
221	0.9	1	1	–	337.3 (337.3)	0.0 (0.0)	0.1 (0.1)	7.0	418.0	90.0	1311.6
241	0.1	5	5	5	480.9 (989.4)	0.0 (0.0)	0.1 (0.7)	3.4	1097.2	12.0	543.9
241	0.3	4	3	–	1655.9 (2791.5)	0.3 (1.1)	0.2 (0.3)	11.2	2823.8	34.1	3970.2
241	0.5	5	3	–	380.7 (658.4)	0.8 (2.4)	0.2 (0.4)	76.2	2155.0	54.8	5060.5
241	0.7	2	0	–	261.1 (315.1)	8.5 (10.0)	0.1 (0.1)	136.5	3104.0	71.8	14680.7
241	0.9	2	0	–	721.4 (940.6)	2.7 (3.6)	0.1 (0.2)	229.5	1315.5	89.8	15327.3
261	0.1	5	4	3	1163.1 (2503.9)	0.0 (0.0)	0.5 (1.0)	8.0	2073.4	11.6	2287.5
261	0.3	5	4	–	1364.6 (1717.8)	0.4 (0.9)	0.3 (0.6)	30.0	2843.0	33.8	4868.8
261	0.5	3	2	–	530.0 (703.1)	5.3 (16.0)	0.2 (0.4)	59.7	1966.7	55.4	6163.6
261	0.7	1	1	–	1112.9 (1112.9)	0.0 (0.0)	0.0 (0.0)	13.0	966.0	73.9	2076.2
261	0.9	3	0	–	651.1 (1012.0)	4.0 (7.3)	0.1 (0.1)	96.0	1245.7	90.3	13406.7
281	0.1	5	5	2	901.9 (1499.1)	0.0 (0.2)	0.5 (1.4)	6.2	1891.4	12.0	1371.4
281	0.3	3	2	–	2176.6 (4802.7)	0.2 (0.7)	0.1 (0.2)	28.3	3013.3	34.8	5685.8
281	0.5	4	2	–	889.1 (987.0)	0.6 (2.1)	0.2 (0.4)	70.0	2773.0	54.4	8543.5
281	0.7	2	0	–	507.5 (708.3)	2.3 (2.9)	0.1 (0.2)	48.5	2076.5	72.1	11518.0
281	0.9	0	0	–	—	—	—	—	—	—	—
301	0.1	5	4	1	1358.1 (1550.8)	0.3 (1.0)	1.1 (2.0)	17.0	2982.8	11.7	3729.6
301	0.3	2	0	–	914.5 (1193.1)	1.5 (2.6)	0.6 (0.8)	98.0	3779.0	35.9	12145.3
301	0.5	3	1	–	530.2 (803.2)	0.9 (2.1)	0.2 (0.3)	59.0	3151.0	54.7	8787.9
301	0.7	1	1	–	616.6 (616.6)	0.3 (0.3)	0.0 (0.0)	14.0	1303.0	72.1	5064.7
301	0.9	0	0	–	—	—	—	—	—	—	—

culties when solving instances with $n = 25$ and $\alpha \geq 0.6$, and with $n = 50$ and $\alpha \geq 0.4$. For example, running the Laporte and Martello code on the instances with $n = 25$ required on average 0.1 seconds for $\alpha = 0.2$, 77.2 seconds for $\alpha = 0.4$, more than 2 hours for $\alpha = 0.6$; whereas for $\alpha = 0.8$, no instance was solved within the 5-hour time-limit.

On the whole, the performance of our branch-and-cut code is quite satisfactory. In most cases, the upper and lower bounds computed at the root node are very tight, and a few branchings are needed. We were able to solve to proven optimality all the random instances of Class III (except 1 instance for $n = 250$, 3 instances for $n = 450$, and 1 instance for $n = 500$), and most of the "real-world" instances of Classes I and II. For the instances exceeding the time limit, the computed solution is very close to the optimal one (see column %-UB).

According to Table II, most of the generated constraints are GSECs, 2-matching, logical, and conditional cuts. For some "difficult" instances, a relevant number of cover and path inequalities is also generated.

Additional computational experience has been performed on the class of random instances considered in the recent work by Gendreau, Laporte, and Semet,[11] called Class IV in the sequel. These instances were generated using the original Gendreau–Laporte–Semet code. These instances are similar to those of Class II and Generation 2, but the nodes are generated as random points in the $[0, 100]^2$ square according to a uniform distribution. The corresponding values of ν(TSP) were computed by means of the algorithm of Padberg and Rinaldi.[20] Table VII reports average (maximum) results over 5 random instances belonging to Class IV, with $\alpha = 0.1, 0.3, 0.5, 0.7, 0.9$, and $n = 20, 40, 60, 80, 100, 120, 140, 160, 180, 200, 220, 240, 260, 280, 300$. Column N_3 gives the number of instances successfully solved by the Gendreau–Laporte–Semet code within a time limit of 10,000 SUN Sparc station 1000 CPU seconds. According to Dongarra,[9] our computer is about 1.8 times faster than that used by Gendreau, Laporte, and Semet, hence their time limit corresponds to about 5,555 HP 9000/720 CPU seconds. Columns N_1 and N_2 give the number of instances successfully solved by our code within a time limit of 18,000 and 5,555 HP 9000/720 CPU seconds, respectively. The remaining columns are as in previous tables, and refer to the execution of our code with the 18,000-second time limit. As in [11], averages are computed with respect to the instances solved to proven optimality.

A comparison of columns N_2 and N_3 shows that our code is capable of solving a number of instances substantially larger than in [11], within the 5,555-seconds time limit. Moreover, both lower and upper bounds computed by our method are tighter than those reported in [11]. For the cases in which Gendreau–Laporte–Semet code successfully solved all five instances, our average LB and UB ratios of 0.02% and 0.28%, respectively, compare very favorably with the LB and UB ratios of 3.59% and 1.74%, respectively, reported in [11].

On the whole, the instances of Class IV appear more difficult than those in the previous classes.

Acknowledgments

Work supported by M.U.R.S.T. and C.N.R., Italy, and by the EEC-Human Capital & Mobility Program (contract No. ERBCIP-DCT940623). We thank Gilbert Laporte and Silvano Martello for having kindly provided us with the FORTRAN codes they used in [15], and Frederic Semet for providing us the random OP generator used in [11].

References

1. E. BALAS, 1989. The Prize Collecting Traveling Salesman Problem, *Networks 19*, 621–636.
2. E. BALAS, 1993. The Prize Collecting Traveling Salesman Problem. II. Polyhedral Results. Working paper, GSIA, Carnegie Mellon University, Pittsburgh.
3. E. BALAS, 1993. On the Cycle Polytope of a Directed Graph. Working paper, GSIA, Carnegie Mellon University, Pittsburgh.
4. E. BALAS and G. MARTIN, 1985. ROLL-A-ROUND: Software Package for Scheduling the Rounds of a Rolling Mill. Copyright, Balas and Martin Associates, 104 Maple Heights Road, Pittsburgh, PA.
5. P. BAUER, 1997. The Cycle Polytope: Facets, *Mathematics of Operations Research 22*, 110–145.
6. I.-M. CHAO, B.L. GOLDEN, and E.A. WASIL, 1996. A Fast and Effective Heuristic for the Orienteering Problem, *European Journal of Operational Research 88*, 475–489.
7. N. CHRISTOFIDES, A. MINGOZZI, and P. TOTH, 1979. The Vehicle Routing Problem, in *Combinatorial Optimization*, N. Christofides, A. Mingozzi, P. Toth, and C. Sandi (eds.), Wiley, Chichester, U.K., 315–338.
8. M. DIABY and R. RAMESH, 1995. The Distribution Problem with Carrier Service: A Dual Based Penalty Approach, *ORSA Journal on Computing 7*, 24–35.
9. J.J. DONGARRA, 1996. Performance of Various Computers Using Standard Linear Equations Software. Report CS-89-85, University of Tennessee, Knoxville.
10. M. FISCHETTI and P. TOTH, 1988. An Additive Approach for the Optimal Solution of the Prize-Collecting Traveling Salesman Problem, in *Vehicle Routing: Methods and Studies*, B.L. Golden, A.A. Assad (eds.), North Holland, Amsterdam, 319–343.
11. M. GENDREAU, G. LAPORTE, and F. SEMET, 1995. A Branch-and-Cut Algorithm for the Undirected Selective Traveling Salesman Problem. Working paper CRT-95-80, Centre de recherche sur les transports, Montréal.
12. B.L. GOLDEN, L. LEVY, and R. VOHRA, 1987. The Orienteering Problem, *Naval Research Logistics 34*, 307–318.
13. B.L. GOLDEN, Q. WANG, and L. LIU, 1988. A Multifaceted Heuristic for the Orienteering Problem, *Naval Research Logistics 35*, 359–366.
14. M. JÜNGER, G. REINELT, and G. RINALDI, 1995. The Traveling Salesman Problem, in *Handbooks in Operations Research and Management Science, Network Models*, Vol. 7, M. Ball, T.L. Magnanti, C.L. Monma, and G. Nemhauser (eds.), North Holland, Amsterdam, 225–330.
15. G. LAPORTE and S. MARTELLO, 1990. The Selective Traveling Salesman Problem, *Discrete Applied Mathematics 26*, 193–207.
16. A.C. LEIFER and M.B. ROSENWEIN, 1994. Strong Linear Programming Relaxations for the Orienteering Problem, *European Journal of Operational Research 73*, 517–523.
17. S. MARTELLO and P. TOTH, 1990. *Knapsack Problems: Algorithms and Computer Implementations*, John Wiley & Sons, Chichester, U.K.
18. G.L. NEMHAUSER and L.A. WOLSEY, 1988. *Integer and Combinatorial Optimization*, Wiley, New York.
19. M.W. PADBERG and M.R. RAO, 1982. Odd Minimum Cut-Sets and *b*-Matchings, *Mathematics of Operations Research 7*, 67–80.
20. M.W. PADBERG and G. RINALDI, 1991. A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems, *SIAM Review 33*, 60–100.
21. R. RAMESH and K.M. BROWN, 1991. An Efficient Four-phase Heuristic for the Generalized Orienteering Problem, *Computers & Operations Research 18*, 151–165.
22. R. RAMESH, Y.-S. YOON, and M.H. KARWAN, 1992. An Optimal Algorithm for the Orienteering Tour Problem, *ORSA Journal on Computing 4*, 155–165.
23. G. REINELT, 1991. TSPLIB—A Traveling Salesman Problem Library, *ORSA Journal on Computing 3*, 376–384.
24. D.J. ROSENKRANTZ, R.E. STEARNS, and P.M. LEWIS II, 1977. An Analysis of Several Heuristics for the Traveling Salesman Problem, *SIAM Journal on Computing 6*, 563–581.
25. T. TSILIGIRIDES, 1984. Heuristic Methods Applied to Orienteering, *Journal of the Operational Research Society 35*, 797–809.