The Team Orienteering Problem: Formulations and Branch-Cut and Price*

Marcus Poggi[1], Henrique Viana[1], and Eduardo Uchoa[2]

1 **Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro**
Rio de Janeiro-RJ, Brasil
{poggi, fviana}@inf.puc-rio.br

2 **Departamento de Engenharia de Produção, Universidade Federal Fluminense**
Niterói-RJ, Brasil
uchoa@producao.uff.br

Abstract

The Team Orienteering Problem is a routing problem on a graph with durations associated to the arcs and profits assigned to visiting the vertices. A fixed number of identical vehicles, with a limited total duration for their routes, is given. The total profit gathered by all routes is to be maximized. We devise an extended formulation where edges are indexed by the time they are placed in the route. A new class of inequalities, min cut, and the triangle clique cuts of Pessoa et. al., 2007 are added. The resulting formulation is solved by column generation. Branching is done following the work of Boussier et al. 2007, to which the branch-cut-and-price algorithm here proposed is compared. A few new upper bounds were obtained. Overall the presented approach has shown to be very competitive.

1998 ACM Subject Classification G.1.6 Optimization, Integer Programming

Keywords and phrases Branch-Cut and Price, Team Orienteering Problem, Column Generation

Digital Object Identifier 10.4230/OASIcs.ATMOS.2010.142

1 Introduction

Routing Problems are among the most studied problems in Combinatorial Optimization. Routing problems consider a fleet of vehicles to visit a set of customers. In most versions of this family of problems, all customers have to be visited exactly once. However, in many applications of the real world, there are constraints that force us to choose which customers to visit. The Team Orienteering Problem (TOP) models one of such situations. In the TOP, each customer has an associated profit and the tours have a maximum duration. The choice of customers is made balancing their profits and their contributions for the route duration. Formally, we consider a complete undirected graph $G(V, E)$ where $V = \{0, \ldots, n+1\}$ is the set of vertices and E is the set of edges. Vertex 0 is the starting point and $n+1$ is the ending point of the routes. A nonnegative profit p_i is associated to each vertex i and l_{ij} is a symmetric travel time between vertices i and j. The fleet has m identical vehicles. The objective is to maximize the total reward collected by all the routes, satisfying the time limit L for each route. Not all customers have to be visited. When only one vehicle is considered, we have the Orienteering Problem, OP, which has been shown to be strongly NP-Hard (see Laporte and Martello(1990) [7]), therefore the TOP is also NP-Hard.

* This work was partially supported by CNPq.

10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'10).
Editors: Thomas Erlebach, Marco Lübbecke; pp. 142–155

OpenAccess Series in Informatics
Schloss Dagstuhl Publishing, Germany

The literature on the *Team Orienteering Problem - TOP* is quite recent. It has been proposed by Butt and Cavalier(1994) [3] with the name *Multiple Tour Maximum Collection Problem*. Two years later, the paper by Chao et al.(1996) [4] formally introduced the problem. As noted above, the TOP is a version of the *Orienteering Problem* considering multiple vehicles. *Orienteering Problems* consider that only one vehicle visit the clients. An exact algorithm for the *Orienteering Problem* was proposed in Fischetti et al.(1998) [5]. The first experimental work on the TOP is presented in Chao at al. [4], it generated the currently most used benchmark instances set. Tang and Miller-Hooks [9] proposed a Tabu Search combined with an adaptive memory procedure. Most of the best known solutions for these TOP benchmark instances are found in Archetti et al.(2005) [1]. This last work proposed two versions of Tabu Search and two metaheuristics implementations based on *Variable Neighborhood Search - VNS*. Ke et al. (2008) [6] developed two ant colonies variations. This approach has been able to get competitive results, reducing the overall computational time. More recently, Vansteenwegen et al.(2009) [10] has presented a VNS which obtains results almost as good as the results in [1], but with a reduced computational time. However, in general, the quality of the solutions presented in [1] are still better. Finally, an exact column generation algorithm, a branch-and-price, has been proposed by Boussier et al. [2]. We use these last results as benchmark for comparison.

In this paper we first present integer programming compact formulations based on arc indexed variables. The first formulation is then extended by considering variables that contain information on the duration of the routes. Next, this latter formulation is decomposed to derive an associated column generation formulation where variables represent routes (elementary or not). Valid inequalities on the arc indexed extended variables are recalled, and new valid inequalities on these variables are proposed. These are joined up in a branch-cut-and-price scheme.

We organized the text as follows. Section 2 addresses the compact and the column generation formulations. Section 3 describes in detail the branch-cut-and-price scheme. The following section 4 presents the experimental results obtained. They are compared with the results in [2]. Finally, conclusions are drawn in section 5.

2 Mathematical Formulations

We now present three Integer Programming formulations for the TOP. The first one is equivalent to the one in Vansteenwegen et al.(2009) [10] which uses variables indicating whether a vehicle route uses or not an arc. Indexing on the vehicles is necessary to take care of the duration of the routes. In the second formulation, these arc indexed variables are also indexed on the instant it starts in the route. This allows to avoid indexing on the vehicles, since no variable indicating an arc will finish after the maximum duration will be considered. The first formulation is said to be compact because its number of variables is polynomial and, although the number of constraints is exponential, they can be separated in polynomial time (subtour elimination constraints). The second formulation has a pseudo-polynomial number of variables and therefore is less compact than the first one. Finally, we present a formulation with an exponential number of columns. Each column represents a possible route and the formulation can be seen as a decomposition of the previous one.

The notation used in the formulations considers a directed complete graph with arc set A. Vertex sets are $V = \{0, \ldots, n+1\}$ and $V^- = \{1, \ldots, n\}$, where the latter contains only the customer vertices. The nonnegative profits are denoted by p_v for $v \in V^-$, arc travel times are given by l_a for $a = (i, j) \in A$. The number of identical vehicles is given by m.

2.1 Compact Formulation

This TOP formulation uses binary variables x_a^k to indicate whether arc a is traversed or not by the vehicle k. Binary variables y_v are set to one to indicate the vertex v is visited and to zero otherwise.

$$\max \quad \sum_{v \in V^-} p_v \cdot y_v \tag{1}$$

$$\sum_{k=1}^{m} \sum_{a \in \delta^-(v)} x_a^k - y_v = 0 \quad \forall v \in V^- \tag{2}$$

$$\sum_{k=1}^{m} \sum_{a \in \delta^-(S)} x_a^k - y_v \geq 0 \quad S \subset V \; \forall v \in S \tag{3}$$

$$\sum_{k=1}^{m} x_a^k \leq 1 \qquad \forall a \in A \tag{4}$$

$$\sum_{a \in A} l_a x_a^k \leq L \qquad k = 1, \ldots, m \tag{5}$$

$$\sum_{a \in \delta^+(v_0)} x_a^k = 1 \qquad k = 1, \ldots, m \tag{6}$$

$$\sum_{a \in \delta^-(v_{n+1})} x_a^k = 1 \qquad k = 1, \ldots, m \tag{7}$$

$$y_v \in \{0, 1\} \qquad \forall v \in V^- \tag{8}$$

$$x_a^k \in \{0, 1\} \qquad \forall a \in A \; \forall k = 1, \ldots, m \tag{9}$$

The objective function (1) maximizes the sum of profits associated to the visited vertices. Constraints (2) ensures that a customer is visited once at most by one vehicle. Connectivity of the routes is guaranteed by constraints (3) that indirectly imposes subtour elimination of optimal solutions. The constraint set (4) forbids the use of an arc by two or more routes. The maximum duration of the routes imposed by constraints (5). Constraint sets (6) and (7) force m vehicles to leave from the starting point and return to the ending point.

2.2 Less Compact Formulation

In this formulation, each arc has an extra index l. This index represents the departure time of a vehicle using the arc. Variable x_a^{lk} indicates that vehicle k passes through arc a starting with l units of time consumed. Since each arc can only be used once, it can start at exactly one single duration spent and we can write:

$$x_a^k = \sum_{l=0}^{L} x_a^{lk} \tag{10}$$

In order to take into account the duration associated to the arcs, we modify the original graph as follows. We create an intermediate vertex w_a for each arc $a \in A$. These artificial vertices have a demand associated to them equal to the travel time of arc a in the original graph. For instance, let w_{ij} be an intermediate point on arc $a = (i, j)$. The original arc becomes two new arcs $a_1 = (i, w_{ij})$ and $a_2 = (w_{ij}, j)$. The resulting modified graph has then arc set $A_1 \bigcup A_2$, where $A_1 = \{(i, w_{ij}), (i, j) \in A\}$ and $A_2 = \{(w_{ij}, j), (i, j) \in A\}$. The vertex set is given by $V \bigcup A$. Figure 1 shows the graph transformation. The intermediate vertex (gray point) consumes a demand (time) that corresponds to the travel time of the original arc (i, j). In this case, the travel time of arc (i, j) is 2.

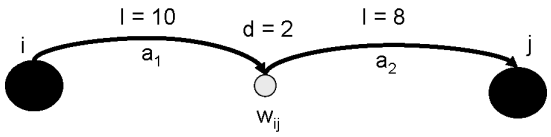

Figure 1 Graph transformation

The formulation on variables x_a^{lk} is obtained by replacing variables x_a^k for equation (10) in the compact formulation 2.1. As mentioned above, constraints (5) can be removed, since the route duration limit can be done by considering only appropriate l indexes. Furthermore, we can impose that m arcs leave vertex 0 and return to vertex $n+1$. This uniquely identify the vehicle routes and, consequently, allow to remove index k from variables x_a^{lk}. The second formulation is then written on variables x_a^l, $a \in A_1 \bigcup A_2$, $l = 0, 1, \ldots, L$. In fact, it would be more precise to use as largest value of l as L minus the travel time of the arc connecting the arrival vertex to vertex $n+1$. We use L to simplify the notation.

$$\max \quad \sum_{v \in V^-} p_v \cdot y_v \tag{11}$$

$$\sum_{l=0}^{L} \sum_{a \in \delta^-(v)} x_a^l - y_v = 0 \qquad \forall v \in V^- \tag{12}$$

$$\sum_{a \in \delta^-(v)} x_a^l - \sum_{a \in \delta^+(v)} x_a^l = 0 \quad \forall l = 0, ..., L \; \forall v \in V^- \tag{13}$$

$$x^l_{(i,w_{ij})} = x^{(l-l_{(i,j)})}_{(w_{ij},j)} \qquad \forall l = 0, ..., L \; \forall (i,j) \in A \tag{14}$$

$$\sum_{l=0}^{L} \sum_{a \in \delta^+(0)} x_a^l = m \tag{15}$$

$$\sum_{a \in \delta^-(n+1)} x_a^0 = m \tag{16}$$

$$y_v \in \{0,1\} \qquad \forall v \in V^- \tag{17}$$

$$x_a^l \in \{0,1\} \qquad \forall a \in A_1 \bigcup A_2 \;\; \forall l = 0, ..., L \tag{18}$$

This formulation can be seen as a flow formulation. Flow conservation is assured by constraints (13) on the original vertices, while constraints (14) do the same on the intermediate vertices. They also impose that traversing an arc consumes time. Finally, constraints (15) and (16) impose the number of routes.

2.3 Column Generation Formulation

As the (pseudo-polynomial) number of variables in the formulation just above may be huge, we apply a Dantzig-wolfe decomposition. The master problem considers only the constraints that keep track of the visited vertices and the one that guarantees the number of routes to be m. The columns represent the routes, therefore assuring the flow conservation constraints are satisfied. For integer solutions of the second formulation the columns are elementary routes. On the other hand, when the linear relaxation is considered, the less compact formulation is equivalent to this column generation formulation when the routes can also be non-elementary or walks on the graph.

All possible columns can be expressed in terms of its arcs indexed by their start instant in the route, elementary or not. Coefficient g_a^{lj} indicates that arc a initiating at duration l is used in route j. Let Q represent the set of all possible routes. Let also λ_j represent the variable indicating whether route (or non-elementary route) j is chosen. We can write:

$$\sum_{j \in Q} g_a^{lj} \cdot \lambda_j = x_a^l \quad \forall a \in A \quad \forall l = 0, ..., L \tag{19}$$

The column generation formulation is then obtained by replacing variable x_a^l in constraints (12) and (16) following the equation above. The formulation is given by:

$$\max \quad \sum_{v \in V^-} p_v \cdot y_v \tag{20}$$

$$\sum_{a \in \delta^-(v)} \sum_{j \in Q} \sum_{l=0}^{L} g_a^{lj} \cdot \lambda_j = y_v \quad \forall v \in V^- \tag{21}$$

$$\sum_{a \in \delta^+(0)} \sum_{j \in Q} g_a^{0j} \cdot \lambda_j = m \tag{22}$$

$$y_v \in \{0, 1\} \quad \forall v \in V^- \tag{23}$$

$$\lambda_j \in \{0, 1\} \quad \forall j \in Q \tag{24}$$

Constraints (21) guarantee that if a vertex is visited, some selected route must visit it. The constraint (22) forces that m routes leave from the starting point.

3 Robust Branch-Cut-and-Price Algorithm

This section describes the proposed Branch-Cut-and-Price algorithm. We first present the pricing subproblem to solve the linear relaxation of the second formulation above by column generation. This implies allowing non-elementary routes to be obtained, but also to avoid solving a strongly NP-Hard problem. The resulting pricing can be solved in pseudo-polynomial time. We say that the Branch-Cut-and-Price is robust regarding its efficiency when this complexity is kept for all pricing done in the algorithm. To preserve this property the cuts presented in following subsections are defined over the variables of the second formulation. Branching is also done keeping this property and is detailed at the end of this section.

3.1 Pricing Subproblem

The pricing subproblem corresponds to finding routes, elementary or not, with maximum reduced cost and duration at most L. This can be done by dynamic programming using the recursion given below (25).

$$r_c^l(j) = max\{r_c^l(j), r_c(i)^{l+l_{(i,j)}} + p(j) - \pi_{(i,j)}\} \tag{25}$$

The maximum reduced cost at vertex j and route duration l is given by $r_c^l(j)$. The value $\pi(i,j)$ is the dual cost of arc (i,j) and sums up all dual contributions from the current restrcited master problem. In the root node of the branch-cut-and-price $\pi(i,j)$ corresponds to the dual variable associated to the j^{th} constraint (21). As cuts are added and branching is done other dual values will contribute to the value of $\pi(i,j)$.

The route, elementary or not, with largest reduced cost is found, regardless of the values of $\pi(i,j)$, in $O(nL)$. It is possible to eliminate 2-cycles in the routes ((i,j) and (j,i) in sequence) without changing this complexity.

3.2 Families of Cuts

Two families of cuts are used in the proposed branch-cut-and price algorithm. The Min Cut inequalities next described is, to the best of our knowledge, new. The second one is an adaptation of the Triangle Clique cuts in Pessoa et al. (2007) [8]. Both are described on variables x^l_a and y_v from the second formulation, the original variables of the column generation formulation. However, the resulting constraints are written in terms of λ_j variables, via equation (19), and added to the column generation formulation.

3.2.1 Min Cut Inequalities

This family of cuts relies on the intuition that fractional solutions for the second formulation will go in and out of a vertex several times, while integer solutions will at most have one value of $x^l_{(i,j)}$ for all i and all l, greater than zero, and equal to one. In a certain sense, it works as a sub-cycle elimination constraint, although it considers all routes with non-negative value in the current solution at once.

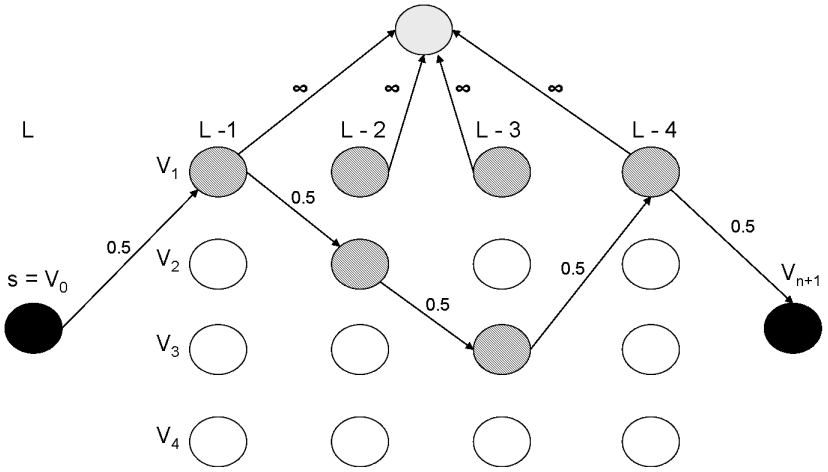

Figure 2 Fractional solution violating a min-cut inequality

Figure 2 presents one such situation where a fractional can be cut off. First observe that exactly one unit of flow enters vertex V_1, satisfying constraints (12), but violating integrality. Now, verify that the minimum cut from the departing vertex V_0 to all copies of V_1 (or to the converging vertex on the top of the figure) is 0.5. This cut is given by the set of all vertices in gray and the minimum value for it is one (or the current value of variable y_1).

Let S be the set of vertices associated to one such minimum cut regarding vertex v. The corresponding inequality is given by:

$$\sum_{a,l|a\in\delta^-(S)} x^l_a \geq y_v \tag{26}$$

Identifying a violated min-cut inequality amounts, therefore, to solve a minimum s-t cut problem. Consider the graph with vertex set $\{0, n+1\} \bigcup \{(i,l)|i \in V^-, \quad l = 0,\ldots,L\}$ and arc set $\{(i^l, j^{l-l_{i,j}})|x^l_{(i,j)} > 0\}$, where the capacity of the arcs are given by the values of variables $x^l_{(i,j)}$ in the current fractional solution. To this graph, add a sink vertex and arcs

from all copies of a vertex v, v^l for $l = 0, \ldots, L$, and assign an infinity capacity. To obtain minimum s-t cuts, we solve max-flow problems on this graph with vertex 0 as source and the required artificial vertex as sink.

The resulting inequality (26) is defined only on variables from the second formulation. Therefore, the associated dual variable will allow assigning its value to the arcs dual costs and the pricing problem will remain unchanged.

3.2.2 Triangle Clique Cuts

Let $S \in V^-$ be a set of exactly three vertices. Consider now all arcs in $A_1 \bigcup A_2$ that has as extreme point on a vertex in S, and their multiplicities on l. Two such arcs are compatible when there exists a route that contains both.

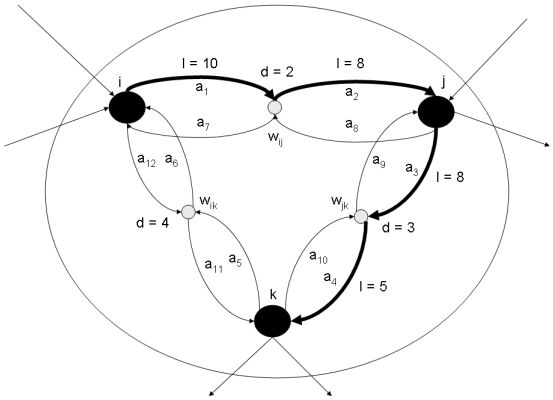

Figure 3 Compatible arcs

Figure 3 illustrate the compatible arcs idea. There are black and gray vertices. The black ones are the vertices of S. The gray ones are the intermediate ones. The index l represents the departure time for each arc. The demand d value on the gray vertices correspond to the travel time of the original arc associated to this intermediate vertex. It can be observed that the arcs in bold describe a possible part of a route and therefore are *compatible*. It worths mentioning that a_5 and a_6 are not compatible with the arcs in bold because if there were a flow returning to vertex i, the arrival time at i would not be equal to 10.

The triangle clique cuts simply states that the sum of the variables associated to arcs (and their multiplicities) in a set where every pair is not compatible can be at most one. This can be view as a clique in an incompatibility graph where there is an edge uniting every pair of incompatible arcs. Another way to look at this same structure is to consider stable sets in a compatibility graph which, in this case, is much less dense.

Let $G' = (V', E')$ be the compatibility graph where each vertex of V' is a time-indexed arc $a^l = (i, j)^l$ for $a \in A_1 \bigcup A_2$ and $l = 0, \ldots, L$. In this case, an edge $e = (a_1^{l_1}, a_2^{l_2})$ belongs to E' if, and only if $a_1^{l_1}$ and $a_2^{l_2}$ are *compatible*. Let $S = \{i, j, k\}$. There are four cases:

Case 1: if $e = ((i, w_{ij})_1^l, (i, w_{ik})_2^l)$, then $e \notin E'$

Case 2: if $e = ((i, w_{ij})_1^l, (k, w_{kj})_2^l)$, then $e \notin E'$

Case 3: if $e = ((i, w_{ij})_1^l, (w_{ij}, k)_2^l)$, and $l_1 \neq l_2 - l(w_{ij})$, then $e \notin E'$

Case 4: if $e = ((i, w_{ij})_1^l, (w_{ij}, k)_2^l)$, and $l_1 = l_2 - l(w_{ij})$, then $e \in E'$

For any stable set $I \subset V'$, the following inequality is valid: $\sum_{a^l \in I} x_a^l \leq 1$

The separation routine for the triangle clique cuts finds the stable set $I \subset V'$ in G' that maximizes $\sum_{a^l \in I} \overline{x}_a^l$, where $\overline{x}_a^l$ denotes the current LP optimal solution. Despite of the problem of finding the maximum-weighted stable set being strongly NP-Hard, we can explore the specific structure of G' and find a maximum weighted independent set in linear time.

A set I is a maximum-weight stable set for a set of chains if, and only if, it is the union of maximum-weight stable set for each single chain. We find in linear time the maximum-weight stable set for each chain H, using a dynamic programming procedure. Let $a_i^{l_i}$ be the ith vertex in the chain H, numbered from 1 to $|H|$ from one extreme to the other of the chain. Let us define $I^*(i,1)$ as the maximum stable set for the subchain containing the first i vertices of H than t does use the i^{th} vertex. Finally, let $c(I) = \sum_{a^l \in I} \overline{x}_a^l$. We have the following recurrence:

$$c(I^*(i,1)) = \overline{x}_{a_i}^{l_i} + c(I^*(i-1,0))$$
$$c(I^*(i,0)) = max(c(I^*(i-1,0)), c(I^*(i-1,1)))$$

It is worth mentioning that these cuts are also a way to eliminate cycles of fixed size in the solution of the restricted master problem.

3.3 Details of the Branch-and-Bound

The branch-cut-and-price algorithm starts with a column generation phase. Once an optimal LP solution is found either cuts are separated or branching is performed. In both cases, the pricing problem must be solved again until another optimal LP solution is obtained.

We branch on the vertices, as in Boussier et al. (2007) [2], deciding whether they are served or not. It is a robust branching scheme because it does not affect the pricing subproblem. Bounding is done using the values of the feasible solutions found in [2].

The master formulation used is a linear relaxation from formulation presented in 2.3. Whenever we fix any variable $\overline{y_v} = 1$, in a node of branch-and-bound tree, there must be a route that visits v. When this is not the case, the problem becomes infeasible. Therefore, to branch on the $\overline{y_v}$ variables, it is necessary to add artificial slack variables $\overline{f^+}$, $\overline{f^-}$ and q, with large costs, to the constraints in order to guarantee the feasibility of the current restricted master problem. Its modified formulation can be written:

$$\max \quad \sum_{v \in V^-} p_v \cdot y_v - \sum_{v \in V^-} M \cdot f_v^+ - \sum_{v \in V^-} M \cdot f_v^- - M \cdot q \tag{27}$$

$$\sum_{a \in \delta^-(v)} \sum_{j \in Q} \sum_{l=0}^{L} g_a^{lj} \cdot \lambda_j + f_v^+ - f_v^- = y_v \qquad \forall v \in V^- \tag{28}$$

$$\sum_{a \in \delta(0)^+} \sum_{j \in Q} g_a^{0j} \cdot \lambda_j + q = m \tag{29}$$

Infeasibility is detected when an artificial slack variable has positive value when LP optimality is reached.

Branching only on the y_v variables may end up with a fractional solution. In this case, we may proceed branching on the x_a^l variables until an integer optimal solution is reached. No results with second branching is presented and when there is still a fractional solution the corresponding upper bound is reported. We choose the y_v variable with value closer to 0.5 to branch on.

4 Computational Experiments

We have tested our algorithm using the instances from Chao et al. (1996) [4]. There are seven datasets where the number of vertices ranges from 21 to 102. For a given number of vertices, instances only differ on the values of L and m. All experiments were performed on a notebook with processor Intel Core Duo (but using a single core) with a clock of 1.66GHz and 2GB of RAM. Results are presented in Tables 1 to 4. All tables have the following columns. *Instance* is the name of the instance file; m is the number of vehicles; L is the maximum duration for the routes; *LB* is the best lower bound, i.e. the value of the best known solution for the TOP instance; *CG* contains the linear relaxation value for the column generation formulation; *ROOT UB* presents the LP upper bound in the root node when both families of cuts are separated; *Our UB* is the value of the best upper bound found by our branch-cut-and-price algorithm; *Boussier UB* is the upper bound presented in Boussier et al.(2007) [2]; columns *CGT*, *CT* and *OT* present the CPU time spent in the pricing problem, in the cut separation procedures and in solving the linear programming problems with CPLEX 11.2, respectively; *NN* indicates the number of nodes explored in the branch-cut-and-price; finally, the *IS* the value of the best integer solution our algorithm found whenever this was the case.

We concentrate our analysis on tables 1 and 2. This is so since the instances in tables 3 and 4 appear to be easy. In those tables, with instances with 64, 66 and 102 vertices, the important remark is that when the column generation formulation did not find the optimal solution value as upper bound, the cut separation lead to this. Moreover, the integer optimal solution was found by our algorithm in 16 out of the 22 instances. Also, the total CPU times where consistently below 2 minutes. This was also the case for the branch-and-price of Boussier et. al. (2007).

Table 1 presents the results for instances with 33 vertices. In the case with 4 vehicles the bounds where identical to the one of Boussier et. al. (2007), with only one instance with a bound above the optimal solution value. In the cases with 2 and 3 vehicles our algorithm compared favorably, obtaining better bounds in 7 out of 9 instances. Again the cut separation improved significantly the bounds. Although a specific column in the tables with only the Min Cut inequalities was not presented, we observed that these were the most effective cuts. Finally, the results on the instances with 100 vertices presented in table 2 can be verified to be similar. In the instances where a tie with the branch-and-price of Boussier et. al. (2007) did not occur, our algorithm outperform theirs in 5 out 6 instances. This was specially true for the most difficult instances p4.4.j and p4.4.k. Again, cut separation was crucial.

5 Conclusions

This work proposes a robust branch-cut-and-price algorithm for the TOP. The experimental results showed that the CPU times were considerably small, even though the duration of the routes were considered with a precision of two decimals, corresponding to large values. These large values explain why solving the pricing subproblem were the most time consuming step. The family of valid inequalities, Min Cut, appeared to be the main contribution of this work, this is so since they can be adapted to a wide class of routing problems. Regarding the TOP, the effort is now on testing and polishing the code with branching on the arc variables. This shall allow finding optimal solutions and to prove their optimality for many of the instances from Chao et al.(1996). To conclude we believe that the ideas here presented allow improving the state-of-the-art in solving instances of the TOP in the near future.

Table 1 Results for instances with 33 vertices

Instance	m	L	LB	CG	ROOT UB	Our UB	Boussier UB	CGT	CT	OT	NN	IS
p3.2.h	2	25	410	430.645	**410**	**410**	417.5	26.56s	18.98s	0.38s	1	410
p3.2.k	2	32.5	550	575.088	**550**	**550**	566.667	81.24s	28.14s	1.17s	1	-
p3.3.e	3	11.7	200	213.333	213.333	210	**200**	2.68s	15.49s	0.32s	4	-
p3.3.i	3	18.3	330	355	335	**330**	336.667	22.00s	17.95s	0.64s	3	330
p3.3.j	3	20	380	403.333	**380**	**380**	390	15.68s	21.50s	0.47s	1	-
p3.3.k	3	21.7	440	458.636	**440**	**440**	450	11.71s	6.11s	0.32s	1	440
p3.3.l	3	23.3	480	503.333	486.667	486.667	**480**	90.17s	86.43s	1.63s	4	-
p3.3.m	3	25	520	537.5	**520**	**520**	526.667	32.10s	27.74s	0.47s	1	-
p3.2.e	2	17.5	260	276.25	**260**	**260**	262	4.96s	0.12s	0.14s	1	260
p3.4.k	4	16.2	350	350	**350**	**350**	**350**	2.47s	0.00s	0.02s	1	350
p3.4.l	4	17.5	380	395	**380**	**380**	**380**	11.89s	0.41s	0.15s	1	-
p3.4.m	4	18.8	390	405	**390**	**390**	**390**	9.51s	0.34s	0.06s	1	-
p3.4.n	4	20	440	461.667	**446.667**	**446.667**	**446.667**	56.59s	4.70s	0.54s	4	-
p3.4.o	4	21.2	500	511.111	**500**	**500**	**500**	14.85s	0.60s	0.13s	1	-
p3.4.p	4	22.5	560	566.667	**560**	**560**	**560**	18.09s	0.58s	0.14s	1	560

Table 2 Results for instances with 100 vertices

Instance	m	L	LB	CG	ROOT UB	Our UB	Boussier UB	CGT	CT	OT	NN	IS
p4.2.a	2	25	206	206	**206**	**206**	**206**	7.47s	0.00s	0.17s	1	206
p4.2.b	2	30	341	344	344	344	**341**	24.70s	2.06s	0.20s	1	-
p4.3.a	3	16.7	0	0	**0**	**0**	**0**	0.15s	0.00s	0.02s	1	-
p4.3.b	3	20	38	38	**38**	**38**	**38**	0.13s	0.00s	0.03s	1	38
p4.3.d	3	26.7	335	342	342	**336.857**	339	136.68s	164.57s	1.95s	12	-
p4.3.e	3	30	468	470.091	469.5	**468.333**	468.75	329.04s	57.52s	2.12s	8	-
p4.3.f	3	33.3	579	591	580.333	**580**	584.5	1020.08s	244.82s	4.36s	8	-
p4.4.d	4	20	38	38	**38**	**38**	**38**	0.12s	0.00s	0.04s	1	38
p4.4.e	4	22.5	183	183	**183**	**183**	**183**	0.62s	0.00s	0.09s	1	-
p4.4.f	4	25	324	324	**324**	**324**	**324**	3.68s	0.00s	0.17s	1	324
p4.4.j	4	35	732	749.41	734.797	**733.38**	741.472	1313.60s	232.69s	7.22s	6	-
p4.4.k	4	37.5	821	841.799	**821.462**	**821.462**	831.945	1937.08s	143.58s	12.29s	4	-

Table 3 Results for instances with 66 vertices

Instance	m	L	LB	CG	ROOT UB	Our UB	Boussier UB	CGT	CT	OT	NN	IS
p5.2.b	2	5	20	20	**20**	**20**	**20**	0.20s	0.00s	0.65s	1	20
p5.2.c	2	7.5	50	50	**50**	**50**	**50**	0.26s	0.00s	2.32s	1	50
p5.2.d	2	10	80	80	**80**	**80**	**80**	0.34s	0.00s	0.31s	1	80
p5.2.e	2	12.5	180	180	**180**	**180**	**180**	1.38s	0.00s	0.14s	1	180
p5.2.f	2	15	240	240	**240**	**240**	**240**	3.79s	0.00s	0.19s	1	240
p5.2.g	2	17.5	320	320	**320**	**320**	**320**	9.88s	0.00s	0.23s	1	320
p5.3.m	3	21.7	650	650	**650**	**650**	**650**	24.46s	0.00s	0.23s	1	650
p5.3.n	3	23.3	755	755	**755**	**755**	**755**	39.90s	0.00s	0.24s	1	-
p5.3.o	3	25	870	870	**870**	**870**	**870**	62.97s	0.00s	0.27s	1	870
p5.3.p	3	26.7	990	990	**990**	**990**	**990**	61.61s	0.00s	0.29s	1	990
p5.4.t	4	25	1160	1160	**1160**	**1160**	**1160**	58.08s	0.00s	0.29s	1	1160
p5.4.u	4	26.2	1300	1300	**1300**	**1300**	**1300**	60.34s	0.00s	0.28s	1	1300
p5.4.v	4	27.5	1320	1320	**1320**	**1320**	**1320**	91.85s	0.00s	0.40s	1	-

Table 4 Results for instances with 64 and 102 vertices

Instance	m	L	LB	CG	ROOT UB	Our UB	Boussier UB	CGT	CT	OT	NN	IS
p6.2.f	2	20	588	588	**588**	**588**	**588**	8.38s	0.00s	0.25s	1	588
p6.2.g	2	22.5	660	660	**660**	**660**	**660**	19.34s	0.00s	0.13s	1	660
p6.4.j	4	15	366	366	**366**	**366**	**366**	0.75s	0.00s	0.02s	1	-
p6.4.k	4	16.2	528	528	**528**	**528**	**528**	1.52s	0.00s	0.01s	1	-
p6.4.n	4	20	1068	1068	**1068**	**1068**	**1068**	26.03s	0.00s	0.05s	1	1068
p7.3.f	3	40	247	247	**247**	**247**	**247**	8.43s	0.00s	0.08s	1	-
p7.3.g	3	46.7	344	349	**344**	**344**	**344**	30.30s	0.14s	0.05s	1	344
p7.4.i	4	45	366	369	**366**	**366**	**366**	22.38s	0.17s	0.05s	1	366
p7.4.j	4	50	462	476.192	**462**	**462**	**462**	50.16s	0.34s	0.06s	1	-

References

1. C Archetti, A Hertz, and M G Speranza. Metaheuristics for the team orienteering problem. *Journal of Heuristics*, 13:49–76, 2005.
2. S Boussier, D Feillet, and M Gendreau. An exact algorithm for team orienteering problems. *4OR*, 5(3):211–230, 2007.
3. S E Butt and T M Cavalier. A heuristic for the multiple tour maximum collection problem. *Computers and Operations Research*, 21:101–111, 1994.
4. I M Chao, B Golden, and E A Wasil. The team orienteering problem. *European Journal of Operational Research*, 88:474–474, 1996.
5. M Fischetti, J Salazar, and P Toth. Solving orienteering problem through branch-and-cut. *INFORMS Journal on Computing*, 10:133–148, 1998.
6. L Ke, C Archetti, and Z Feng. Ants can solve the team orienteering problem. *Computers and Industrial Engeneering*, 54:648–665, 2008.
7. G Laporte and S Martello. The selective traveling salesman problem. *Discrete Appl Math*, 26:193–207, 1990.
8. A Pessoa, M Poggi, and E Uchoa. A robust branch-cut-and-price algorithm for the heterogeneous fleet vehicle routing problem. *Networks*, 54:167–177, 2009.
9. H. Tang and E. Miller-Hooks. A tabu search heuristic for the team orienteering problem. *Computers and Operations Research*, 32:1379–1407, 2005.
10. P Vansteenwegen, W Souffriou, G Vanden Berghe, and D Van Oudheusden. A guided local search metaheuristic for the team orienteering problem. *European Journal of Operational Research*, 196(1):118–127, 2009.