4OR (2007) 5:211–230
DOI 10.1007/s10288-006-0009-1

REGULAR PAPER

An exact algorithm for team orienteering problems

Sylvain Boussier · Dominique Feillet · Michel Gendreau

Received: 1 November 2005 / Revised: 1 May 2006 / Published online: 18 July 2006

Abstract Optimising routing of vehicles constitutes a major logistic stake in many industrial contexts. We are interested here in the optimal resolution of special cases of vehicle routing problems, known as team orienteering problems. In these problems, vehicles are guided by a reward that can be collected from customers, while the length of routes is limited. The main difference with classical vehicle routing problems is that not all customers have to be visited. The solution method we propose here is based on a Branch & Price algorithm. It is, as far as we know, the first exact method proposed for such problems, except for a preliminary work from Gueguen (Methodes de résolution exacte pour problémes de tournées de véhicules. Thése de doctorat, école Centrale Paris 1999) and a work from Butt and Ryan (Comput Oper Res 26(4):427–441 1999). It permits to solve instances with up to 100 customers.

Keywords Selective vehicle routing problem with time windows · Team orienteering problem · Column generation · Branch & price · Routing problems with profits

S. Boussier (✉)
LGI2P, Ecole des Mines d'Alès - Site EERIE, Parc Scientifique Georges Besse,
30035 Nimes Cedex 1, France
e-mail: Sylvain.Boussier@ema.fr

D. Feillet
Laboratoire d'informatique d'Avignon, Université d'Avignon et des pays de Vaucluse,
339 Chemin des Meinajaries, Agroparc B.P. 1228, Avignon Cedex 9, France
e-mail: dominique.feillet@univ-avignon.fr

M. Gendreau
Centre de recherche sur les transports, Université de Montréal,
6128 succursale centre-ville, Montréal, Canada H3C 3J7
e-mail: michelg@crt.umontreal.ca

MSC Classification 90C10 · 90C35 · 90C90

1 Introduction

Optimising routing of vehicles constitutes a major logistic stake in many industrial contexts. We are interested here in the optimal resolution of special cases of vehicle routing problems, known as team orienteering problems. In these problems, a set of potential customers is considered and a fleet of vehicles is used to visit a part of these customers. The visit of a customer provides a given reward. Because of a limitation on vehicle autonomy, the length of routes is constrained. The objective is to construct vehicle routes such that the total reward received from visited customers is maximized. The main difference with classical vehicle routing problems is that not all customers have to be visited. A larger class of problems, including team orienteering problems, has been called routing problems with profits in Feillet et al. (2005a).

We propose here a method based on a Branch & Price algorithm for solving team orienteering problems. Such approach has the advantage to be easily adaptable to different sorts of team orienteering problems. We evaluate it on two particular problems. The first one is widely known as the team orienteering problem (TOP). The second one is coined as the selective vehicle routing problem with time windows (SVRPTW) by Gueguen (1999) and Hayari et al. (2003). These two problems are extensions of the orienteering problem (OP, also known as the selective traveling salesman problem) to a multivehicle situation. The OP consists of, given a graph with profits associated to nodes, find a path of maximum profit from a given origin to a given destination node, such that the length of the path does not exceed a given limit. Laporte and Martello (1990) show that the OP is NP-hard. In the TOP, one searches for k separated paths of limited length. It is first introduced by Chao et al. (1996). Several heuristic approaches have been proposed for its solution, as, recently, Tang and Miller-Hooks (2005) and Archetti et al. (2005). The SVRPTW introduces additional capacity and time window constraints.

As far as we know, the only exact solution approaches devoted to team orienteering problems are proposed by Gueguen (1999) for the SVRPTW and by Butt and Ryan (1999) for a variant of the TOP introducing heterogeneous vehicles, named the multiple tour maximum collection problem. Both approaches rely on similar set covering formulations, solved with Branch & Price, but with quite different subproblem resolution and branching strategies. More details on these algorithms can be found in Feillet et al. (2005a). Our attempt in this work is to propose a generic Branch & Price scheme capable of solving efficiently different kinds of orienteering problems. We essentially continue the preliminary work of Gueguen (1999), with new branching strategies and several acceleration techniques. Butt and Ryan (1999) approach is quite much specific to the multiple tour maximum collection problem and cannot easily handle specific constraints.

Section 2 describes our Branch & Price algorithm. Sections 3 and 4 then, respectively, show how it can be applied to the TOP and the SVRPTW, and demonstrate its efficiency.

2 A Branch & Price algorithm for team orienteering problems

2.1 Description of team orienteering problems

Team orienteering problems can be defined as follows: Let $G = (V, E)$ be a complete graph, where $V = \{v_0, v_1, \ldots, v_{n+1}\}$ is the vertex set and E is the edge set. Vertices v_1 to v_n stand for customers, vertices v_0 and v_{n+1}, respectively, stand for departure and ending points (and might actually correspond to a same physical location). Let l_{ij} be the distance between $v_i \in V$ and $v_j \in V$. Let p_i be the reward (profit) received when visiting vertex $v_i \in V$, with $p_0 = p_{n+1} = 0$. Let m be the number of identical available vehicles and L be a limit on the length of the routes of these vehicles. Team orienteering problems consist in finding a set of m vehicle routes, going from v_0 to v_{n+1}, keeping to the length limitation, such that each customer is visited at most once and that the total collected profit is maximized.

Many additional constraints can be considered depending on the context of application (capacity constraints, time windows…). One major advantage of Branch & Price algorithms is that most of such constraints only affect the subproblem devoted to generate new columns. Furthermore, when this subproblem is solved using dynamic programming, as we propose following Gueguen (1999), these constraints can generally be easily handled. Section 4 will explain how capacity constraints, time windows and service times can be considered in the context of the SVRPTW. A fleet heterogeneity, as introduced by Butt and Ryan (1999), could also be treated by simply considering a subproblem for each type of vehicle. Many other possible constraints (e.g., precedence constraints) are described in Irnich and Desaulniers (2005).

2.2 Formulation

Let $\Omega = \{r_1, \ldots, r_{|\Omega|}\}$ be the set of possible routes for a vehicle, that is, the set of routes originating from v_0, ending at v_{n+1}, visiting at most once each customer and such that the sum of distances of the arcs travelled does not exceed L. Let p_k be the reward generated by route $r_k \in \Omega$. Let $a_{ik} = 1$ if route $r_k \in \Omega$ visits customer v_i and $a_{ik} = 0$ otherwise. Team orienteering problems can be stated as follows:

$$\text{maximize} \sum_{r_k \in \Omega} p_k x_k \tag{1}$$

subject to

$$\sum_{r_k \in \Omega} a_{ik} x_k \leq 1 (v_i \in V \setminus \{v_0 \cup v_{n+1}\}), \tag{2}$$

$$\sum_{r_k \in \Omega} x_k \leq m, \tag{3}$$

$$x_k \in \{0, 1\} (r_k \in \Omega). \tag{4}$$

In this model, decision variables x_k indicate whether route $r_k \in \Omega$ is used or not. Constraints (2) ensure that each customer is visited at most once. Constraint (3) limits the number of used vehicles to m.

Solving the linear relaxation of model (1)–(4) necessitates the use of a column generation technique, due to the size of Ω. Coupling column generation with Branch & Bound then allows the resolution of (1)–(4). Such scheme is called Branch & Price. In the following, we call Master Problem (MP) the linear relaxation of model (1)–(4).

2.3 Column generation phase

Column generation is based on two components: a restricted master problem and a subproblem. The restricted master problem RMP(Ω_1) is obtained from MP by considering only a subset $\Omega_1 \subset \Omega$ of variables. The subproblem aims at adding progressively new potentially good columns to Ω_1 until an optimality criterion is attained. The reader is referred to Desaulniers et al. (2005) for a recent book on the subject.

The Ω_1 is initialized with a simple set of routes, each visiting a single customer. At each iteration of the algorithm, RMP(Ω_1) is solved with the simplex method, and provides optimal dual variables. Denoting by λ_i the nonnegative dual variable associated with constraint (2) for customer v_i and λ_0 the nonnegative dual variable associated with constraint (3), the subproblem determines whether some variables x_k with $r_k \in \Omega \setminus \Omega_1$ have a positive reduced cost. This condition can easily be stated as:

$$\sum_{v_i \in V \setminus \{v_0 \cup v_{n+1}\}} a_{ik} \lambda_i + \lambda_0 < p_k,$$

One or several variables with positive reduced cost are then added to Ω_1 and the algorithm iterates until the subproblem fails to find new routes.

The condition checked by the supbroblem can equivalently be written as:

$$\sum_{v_i \in r_k} (\lambda_i - p_i) + (\lambda_0 - p_0) < 0,$$

where $v_i \in r_k$ means that v_i is a costumer visited by r_k (note that $p_0 = 0$). This expression illustrates that the subproblem can be apprehended as an Elementary Shortest Path Problem with Resource Constraints (ESPPRC). To be aware of that, one has to consider graph G and to introduce a cost $c_{ij} = \lambda_i - p_i$ for each arc (v_i, v_j). A resource has then to be defined for taking account of the length limit; each arc (v_i, v_j) consumes a quantity l_{ij} of this resource; a feasibility limit is set to L. The path has to be elementary in the sense that customers should not be visited more that once. The subproblem then consists in finding an elementary path, connecting v_0 to v_{n+1}, satisfying the resource constraint, with a negative cost.

Other resources could be added depending on the context. A resource r would be defined by a resource consumption function f_{ij}^r on every arc (v_i, v_j), indicating the resource consumption level $T_j^r = f_{ij}^r(T_i^r)$ of a path arriving at v_i with a level T_i^r and traversing arc (v_i, v_j). Resource constraints will then be imposed by a resource window $\left[a_i^r, b_i^r\right]$ on every node v_i.

The ESPPRC can be solved through a dynamic programming approach, as proposed by Feillet et al. (2004). One should note that the elementary path condition complicates the subproblem and could be removed. However, Ω would then be extended to paths visiting vertices several times, and therefore collecting profits several times, which would strongly weaken the quality of the upper bound provided by MP.

We need a very short description of the algorithm of Feillet et al. (2004) for the subsequent sections. The algorithm is based on dynamic programming. It follows the classical Bellman's algorithm. The principle is to associate a label with each possible partial path and to extend these labels checking the resource constraints, until the best feasible paths are obtained. Dominance rules are used to compare labels and remove some of them.

2.4 Branching phase

In order to obtain the optimal solution of (1)–(4), column generation has to be embedded into a branching scheme. It is well known that branching on x_k variables would heavily complicate the subproblem phase, because of the difficulty to handle forbidden paths. Gueguen (1999) proposes to use a branching rule initially developed for the Vehicle Routing Problem with Time Windows (see, e.g., Desaulniers et al. 2005). This rule relies on the selection of an arc (v_i, v_j) traversed by a fractional quantity of flow, that is a fractional number of vehicles. One can easily see that such an arc necessarily exists when the solution is fractional. The problem is then split into two branches. In the fist branch, routes r_k containing (v_i, v_j) are removed from Ω_1 and (v_i, v_j) is removed from the graph during the subproblem phase. In the second branch, routes r_k using either outgoing arcs from v_i other than (v_i, v_j) or ingoing arc to v_j other than (v_i, v_j) are removed from Ω_1 and these arcs are removed from the graph during the subproblem phase. Unfortunately this branching rule cannot be exactly replicated for team orienteering problems.

A problematical situation would be for example a solution of MP where a route with a fractional value uses an arc (v_i, v_j) such that neither v_i nor v_j belongs to any other route of the solution. By applying the above rule on arc

(v_i, v_j), the solution would still be feasible for MP in the second branch. The algorithm might then diverge. Furthermore, solutions where neither v_i nor v_j are visited belong to both branches, which is correct but would preferably be avoided. For these two reasons, we introduce a different branching rule. This rule introduces two sorts of branchings.

When the solution of MP is fractional, we first search for a customer v_i visited a fractional number of times. Two branches are derived, forbidding or enforcing the visit of v_i. At the master problem level, constraint (2) of RMP(Ω_1) is updated as follows for v_i:

$$\sum_{r_k \in \Omega} a_{ik} x_k = \begin{cases} 1 & \text{in the branch where } v_i \text{ must be served} \\ 0 & \text{in the branch where the visit of } v_i \text{ is forbidden} \end{cases}$$

When v_i is forbidden, v_i is also removed from the graph at the subproblem level. If several customers with a fractional value exist, we select the one with smallest value $\lambda_i - p_i$. This rule should penalize the branch where the vertex is forbidden, which is potentially more difficult to solve.

When the flow traversing each customer is integer, we base our branching on an arc (v_i, v_j) with a fractional flow. Two cases can be considered:

1. If v_i or v_j is constrained to be served, we derive two branches, as in Gueguen (1999): one branch where (v_i, v_j) is forbidden, one branch where it is enforced. The condition that v_i or v_j is traversed with a flow 1 permits to avoid the situation described above.
2. If neither v_i nor v_j is constrained to be served, we derive three branches. The first branch forbids v_i. The second branch enforces the visit of v_i and obliges the use of (v_i, v_j). The third branch enforces the visit of v_i and forbid the use of (v_i, v_j). In each case, the flow on (v_i, v_j) is going to be integer.

Branching constraints on arcs and vertices are handled as described above. However, we still have a difficulty to face out. Constraints might render RMP(Ω_1) infeasible. We tackle this situation by adding a virtual variable z in the formulation. This variable is added with a large negative value in the objective function. It also appears in constraints imposing the service of customers, that become $\sum_{r_k \in \Omega} a_{ik} x_k + z = 1$ for a customer v_i.

2.5 Acceleration techniques

First, in order to increase the performance of the algorithm, we stop the subproblem prematurely when a large set of good columns has been found. We define this limit as 500 routes with a negative cost. This limit avoids spending too much time solving subproblems optimally, while we do not necessarily need the route with the optimal reduced cost value. Besides this strategy, we implement and adapt to the orienteering context two other acceleration techniques proposed by Feillet et al. (2005b), which we describe next.

2.5.1 Limited discrepancy search

The limited discrepancy search (LDS) is a heuristic tree-search method introduced by Harvey and Ginsberg (1995) within a constraint programming paradigm. The principle of LDS is to visit only promising nodes, the interest of a node being defined with a heuristic rule, with the exception that a limited number of discrepancies are allowed. In our dynamic programming algorithm, each vertex of the graph is allotted a limited number of good neighbours. A level of discrepancy is associated with each label. A discrepancy is counted for each connection towards a bad neighbour in the partial path represented by the label. Extensions of label are only authorized when they do not cause exceeding a discrepancy limit DISCMAX.

Good neighbours of a vertex v_i are defined according to two criteria. The first criterion highlights the two vertices v_j with lowest values $\lambda_j - p_j + l_{ij}$. The second criterion highlights vertices v_j belonging to the set of visited vertices in the father node of the Branch & Price tree. This second criterion is not applied at the root node. This criterion results from the empirical observation that the set of visited vertices is very stable during the search. A note is attributed to each neighbour: O if it is a good vertex according to both criteria, 1 if it is a good neighbour for exactly one criterion, 2 otherwise. This note is then used to count the number of discrepancies for partial paths traversing arc (v_i, v_j).

The DISCMAX is initialized to zero. As long as the subproblem does not find new routes, DISCMAX is increased as detailed in Fig. 1, roughly speaking it doubles. In this figure, MAXPOSSIBLE is the maximum number of vertices a route can contain. This value is computed solving a simple knapsack problem (see Feillet et al. 2005b for details).

2.5.2 Label loading and meta extensions

Label loading and meta extensions are two preprocessing procedures aiming at exploiting information provided by RMP(Ω_1). Both procedures rely on the

```
DISCMAX=0
STOP=false
while STOP = false
    solve the subproblem with maximum discrepancy value DISCMAX
    if the subproblem fails to find new routes
        if DISCMAX = 2×MAXPOSSIBLE
            STOP=true
        if DISCMAX ≥ 1
            DISCMAX = 2×DISCMAX
        if DISCMAX = 0
            DISCMAX = 1
        if DISCMAX > MAXPOSSIBLE
            DISCMAX = 2×MAXPOSSIBLE
    else STOP=true
end while
```

Fig. 1 LDS algorithm for the subproblem resolution property that a route $r_k \in \Omega_1$ such that $x_k > 0$ in the optimal solution of RMP(Ω_1), has a reduced cost value equal to 0, which made it a good starting point for finding good routes. We note R the set of routes satisfying this condition.

The label loading procedure generates a label for each intermediate stage of routes $r_k \in R$. When the dynamic programming algorithm is launched, these labels are soon extended, which drives the search towards routes in the neighbourhood of R.

The meta extensions procedure behaves in an opposite fashion. A meta-vertex is built for each intermediate stage of routes $r_k \in R$. This meta-vertex represents the ending part of the route. When a label is extended towards a meta-vertex, it generates a new label at the destination vertex in one shot. Time windows are defined for each meta-vertex, for each resource, according to the consumption of resources in r_k.

These two procedures can be interpreted as priority rules enhancing the search towards routes of R.

3 Application to the team orienteering problem

Team orienteering is an outdoor sport usually played in a mountainous or heavily forested area. Each member of a competitor team (2, 3 or 4 members) armed with compass and map, starts at a specified control point, tries to visit as many other control points as possible within a prescribed time limit, and returns to a specified control point. Each control point has an associated score, so that the objective of orienteering is to maximize the total score. Once a team member visits a point and is awarded the associated score, no other team member can be awarded a score for visiting the same point. Thus, each member of a team has to select a subset of control points to visit in order to maximize the total reward collected.

Chao et al. (1996) introduce the TOP in this context. It exactly corresponds to the description presented in Sect. 2.1, without any additional constraint. We have evaluated our Branch & Price algorithm for the TOP, on a set of instances proposed by Chao et al. (1996) and used ever since to assess heuristic procedures (Tang and Miller-Hooks 2005, Archetti et al. 2005). The data set contains 387 instances with 7 different values for the number of control points, varying between 21 and 102. For a given number of control points, the only differences between instances are L and m; m varies from 2 to 4; a set of values is defined for

Table 1 TOP – instances

Number of control points	21	32	33	64	66	100	102
Values for L	a-k	a-r	a-t	a-n	a-z	a-t	a-t
Values for m	2-4	2-4	2-4	2-4	2-4	2-4	2-4
Number of instances	33	54	60	42	78	60	60

Table 2 TOP – results for instances with 21 vertices

Instance	m	L	MP		Integer solution			
			Val	CPU(s)	Val	CPU(s)	gap	#
p2.2.a	2	7.5	90	0	90	0	0	8
p2.2.b		10	120	0	120	0	0	10
p2.2.c		11.5	140	0	140	0	0	11
p2.2.d		12.5	160	0	160	0	0	12
p2.2.e		13.5	190	0	190	0	0	13
p2.2.f		15	200	0	200	0	0	14
p2.2.g		16	200	0	200	0	0	14
p2.2.h		17.5	230	0	230	0	0	15
p2.2.i		19	230	0	230	0	0	15
p2.2.j		20	260	1	260	1	0	16
p2.2.k		22.5	284	0	275	0	3.16	16
p2.3.a	3	5	70	0	70	0	0	8
p2.3.b		6.7	70	0	70	0	0	7
p2.3.c		7.7	105	0	105	0	0	10
p2.3.d		8.3	105	0	105	0	0	10
p2.3.e		9	120	0	120	1	0	11
p2.3.f		10	120	0	120	0	0	10
p2.3.g		10.7	145	0	145	0	0	12
p2.3.h		11.7	165	0	165	0	0	13
p2.3.i		12.7	200	0	200	0	0	15
p2.3.j		13.3	200	0	200	0	0	15
p2.3.k		15	200	0	200	0	0	15
p2.4.a	4	3.8	10	0	10	0	0	2
p2.4.b		5	70	0	70	0	0	8
p2.4.c		5.8	70	0	70	0	0	8
p2.4.d		6.2	70	0	70	0	0	7
p2.4.e		6.8	70	0	70	0	0	7
p2.4.f		7.5	105	0	105	0	0	11
p2.4.g		8	105	0	105	0	0	10
p2.4.h		8.8	120	0	120	0	0	12
p2.4.i		9.5	120	0	120	0	0	11
p2.4.j		10	120	0	120	0	0	10
p2.4.k		11.2	180	0	180	0	0	15

L, represented by a letter at the end of the name of the instance; note however that an identical letter does not necessarily corresponds to an identical value for L when the number of control points changes. The set of instances is described in Table 1.

Computational experiments have been carried out on a PC Pentium IV 3.2 GHz, with a time limit of 2 h. Results are given in Tables 2, 3, 4, 5, 6, 7 and 8, where:

- *Instance* is the name of the instance,
- m is the number of team members,
- L is the time limit,
- val is the value of the optimal solution (relaxed or integer),
- CPU(s) is the CPU time in seconds of the resolution,

Table 3 TOP – results for instances with 32 vertices

Instance	m	L	MP		Integer solution			
			Val	CPU(s)	Val	CPU(s)	gap	#
p1.2.a	2	2.5	0	0	0	0	0	0
p1.2.b		5	15	0	15	0	0	4
p1.2.c		7.5	20	0	20	0	0	5
p1.2.d		10	30	0	30	0	0	6
p1.2.e		12.5	45	0	45	0	0	7
p1.2.f		15	80	0	80	0	0	11
p1.2.g		17.5	90	0	90	0	0	12
p1.2.h		20	112.5	0	110	0	2.22	14
p1.2.i		23	135	0	135	1	0	18
p1.2.j		25	157.5	0	155	1	1.58	19
p1.2.k		27.5	176	3	175	12	0.56	22
p1.2.l		30	195	9	195	22	0	23
p1.2.m		32.5	215	44	215	44	0	24
p1.2.n		35	235	794	235	794	0	25
p1.2.o		36.5	240	1,038	240	1038	0	26
p1.2.p		37.5	250	2,926	–	–	–	–
p1.3.a	3	1.7	0	0	0	0	0	0
p1.3.b		3.3	0	0	0	0	0	0
p1.3.c		5	15	0	15	0	0	4
p1.3.d		6.7	15	0	15	0	0	4
p1.3.e		8.3	30	0	30	0	0	7
p1.3.f		10	40	0	40	0	0	9
p1.3.g		11.7	50	0	50	0	0	9
p1.3.h		13.3	72.5	0	70	0	3.44	12
p1.3.i		15.3	105	0	105	0	0	16
p1.3.j		16.7	115	0	115	0	0	17
p1.3.k		18.3	135	0	135	0	0	18
p1.3.l		20	155	0	155	0	0	22
p1.3.m		21.7	177.5	0	175	0	1.40	21
p1.3.n		23.3	191.667	0	190	2	0.86	23
p1.3.o		24.3	206	0	205	1	0.48	24
p1.3.p		25	220	0	220	0	0	26
p1.3.q		26.7	234	2	230	20	1.70	28
p1.3.r		28.3	252.5	4	250	14	0.99	29
p1.4.a	4	1.2	0	0	0	0	0	0
p1.4.b		2.5	0	0	0	0	0	0
p1.4.c		3.8	0	0	0	0	0	0
p1.4.d		5	15	0	15	0	0	4
p1.4.e		6.2	15	0	15	0	0	4
p1.4.f		7.5	25	0	25	0	0	8
p1.4.g		8.8	35	0	35	0	0	9
p1.4.h		10	45	0	45	0	0	11
p1.4.i		11.5	60	0	60	0	0	12
p1.4.j		12.5	75	0	75	0	0	12
p1.4.k		13.8	100	0	100	0	0	16
p1.4.l		15	120	0	120	0	0	18
p1.4.m		16.2	131.667	0	130	0	1.26	20
p1.4.n		17.5	155	0	155	0	0	22
p1.4.o		18.2	165	0	165	0	0	23
p1.4.p		18.8	175	0	175	0	0	24
p1.4.q		20	190	0	190	0	0	26
p1.4.r		21.2	210	1	210	1	0	28

Table 4 TOP – results for instances with 33 vertices

Instance	m	L	MP		Integer solution			
			Val	CPU(s)	Val	CPU(s)	gap	#
p3.2.a	2	7.5	90	0	90	0	0	7
p3.2.b		10	150	0	150	0	0	9
p3.2.c		12.5	180	0	180	0	0	11
p3.2.d		15	223.333	1	220	1	1.49	13
p3.2.e		17.5	262	0	260	0	0.76	14
p3.2.f		20	303.333	0	300	1	1.09	16
p3.2.g		22.5	367.143	1	360	2	1.94	17
p3.2.h		25	417.5	7	410	34	1.79	20
p3.2.i		27.5	465	11	460	33	1.07	21
p3.2.j		30	518	21	510	188	1.54	23
p3.2.k		32.5	566.667	270	550	3,135	2.94	25
p3.2.l		35	605	4,737	–	–	–	–
p3.2.t		55	800	7	800	7	0	33
p3.3.a	3	5	30	0	30	0	0	6
p3.3.b		6.7	90	0	90	0	0	8
p3.3.c		8.3	120	0	120	0	0	9
p3.3.d		10	170	0	170	0	0	11
p3.3.e		11.7	200	0	200	0	0	13
p3.3.f		13.3	230	0	230	0	0	15
p3.3.g		15	273.333	1	270	1	1.21	17
p3.3.h		16.7	300	0	300	0	0	17
p3.3.i		18.3	336.667	0	330	0	1.98	21
p3.3.j		20	390	0	380	1	2.56	21
p3.3.k		21.7	450	1	440	2	2.22	23
p3.3.l		23.3	488	0	480	3	1.63	24
p3.3.m		25	526.667	1	520	13	1.26	25
p3.3.n		26.7	571.667	6	570	13	0.29	27
p3.3.o		28.3	609.804	8	590	1,104	3.24	27
p3.3.p		30	658.182	13	640	468	2.76	29
p3.3.q		31.7	684.137	99	680	167	0.60	31
p3.3.r		33.3	710	1	710	1	0	32
p3.3.s		35	738.913	416	–	–	–	–
p3.3.t		36.7	763.688	4,181	–	–	–	–
p3.4.a	4	3.8	20	0	20	0	0	4
p3.4.b		5	30	0	30	0	0	6
p3.4.c		6.2	90	0	90	0	0	8
p3.4.d		7.5	100	0	100	0	0	9
p3.4.e		8.8	140	0	140	0	0	11
p3.4.f		10	190	0	190	0	0	14
p3.4.g		11.2	220	0	220	0	0	16
p3.4.h		12.5	240	0	240	0	0	17
p3.4.i		13.8	270	0	270	0	0	19
p3.4.j		15	315	0	310	0	1.58	20
p3.4.k		16.2	350	0	350	0	0	21
p3.4.l		17.5	380	0	380	0	0	23
p3.4.m		18.8	390	1	390	1	0	23
p3.4.n		20	446.667	0	440	0	1.49	25
p3.4.o		21.2	500	0	500	0	0	27
p3.4.p		22.5	560	1	560	1	0	29
p3.4.q		23.8	574.667	1	560	10	2.55	29
p3.4.r		25	605	1	600	3	0.82	30
p3.4.s		26.2	670	3	670	3	0	32
p3.4.t		27.5	670	0	670	0	0	32

Table 5 TOP – results for instances with 64 vertices

Instance	m	L	MP		Integer solution			
			Val	CPU(s)	Val	CPU(s)	gap	#
p6.2.a	2	7.5	0	0	0	0	0	0
p6.2.b		10	0	0	0	0	0	0
p6.2.c		12.5	0	0	0	0	0	0
p6.2.d		15	192	0	192	0	0	16
p6.2.e		17.5	360	0	360	0	0	16
p6.2.f		20	588	0	588	0	0	28
p6.2.g		22.5	660	1	660	1	0	30
p6.2.h		25	780	16	780	16	0	34
p6.2.i		27.5	888	1,397	888	1,397	0	38
p6.3.a	3	5	0	0	0	0	0	0
p6.3.b		6.7	0	0	0	0	0	0
p6.3.c		8.3	0	0	0	0	0	0
p6.3.d		10	0	0	0	0	0	0
p6.3.e		11.7	0	0	0	0	0	0
p6.3.f		13.3	0	0	0	0	0	0
p6.3.g		15	282	0	282	0	0	22
p6.3.h		16.7	444	0	444	0	0	25
p6.3.i		18.3	642	1	642	1	0	36
p6.3.j		20	828	1	828	1	0	40
p6.3.k		21.7	936	1	894	3,965	4.48	41
p6.3.l		23.3	1,014	7	1,002	278	1.18	46
p6.3.m		25	1,104	33	–	–	–	–
p6.3.n		26.7	1,170	441	1,170	4,634	0	54
p6.4.a	4	3.8	0	0	0	0	0	0
p6.4.b		5	0	0	0	0	0	0
p6.4.c		6.2	0	0	0	0	0	0
p6.4.d		7.5	0	0	0	0	0	0
p6.4.e		8.8	0	0	0	0	0	0
p6.4.f		10	0	0	0	0	0	0
p6.4.g		11.2	0	0	0	0	0	0
p6.4.h		12.5	0	0	0	0	0	0
p6.4.i		13.8	0	0	0	0	0	0
p6.4.j		15	366	0	366	0	0	27
p6.4.k		16.2	528	0	528	0	0	32
p6.4.l		17.5	708	1	696	3	1.69	41
p6.4.m		18.8	948	1	912	4	3.79	46
p6.4.n		20	1,068	1	1,068	1	0	52

- gap is the gap (in percentage) between the relaxation and the integer solution, i.e., gap$= 100 \times \frac{\text{val(MP)}-\text{val(IntegerSolution)}}{\text{val(MP)}}$,
- # is the number of visited vertices.

These tables show the ability of our algorithm for solving TOP instances. We solve 270 out of the 387 instances. The remaining 117 instances are not solved in 2 h and are not presented in the tables, except when we are able to compute the

Table 6 TOP – results for instances with 66 vertices

Instance	m	L	MP		Integer solution			
			Val	CPU(s)	Val	CPU(s)	gap	#
p5.2.a	2	2.5	0	0	0	0	0	0
p5.2.b		5	20	0	20	0	0	6
p5.2.c		7.5	50	0	50	0	0	8
p5.2.d		10	80	0	80	0	0	10
p5.2.e		12.5	180	1	180	1	0	14
p5.2.f		15	240	0	240	0	0	14
p5.2.g		17.5	320	0	320	0	0	18
p5.2.h		20	410	2	410	2	0	20
p5.2.i		22.5	480	16	480	16	0	22
p5.2.j		25	580	260	580	260	0	22
p5.2.k		27.5	670	4,893	670	4,893	0	28
p5.3.a	3	1.7	0	0	0	0	0	0
p5.3.b		3.3	15	0	15	0	0	6
p5.3.c		5	20	0	20	0	0	7
p5.3.d		6.7	60	0	60	0	0	9
p5.3.e		8.3	95	0	95	0	0	12
p5.3.f		10	110	0	110	0	0	13
p5.3.g		11.7	185	0	185	0	0	16
p5.3.h		13.3	260	0	260	0	0	19
p5.3.i		15	335	1	335	1	0	20
p5.3.j		16.7	470	0	470	0	0	25
p5.3.k		18.3	495	0	495	0	0	24
p5.3.l		20	605	3	595	33	1.65	28
p5.3.m		21.7	650	2	650	2	0	31
p5.3.n		23.3	755	42	755	42	0	34
p5.3.o		25	870	251	870	251	0	33
p5.3.p		26.7	990	2,258	990	2,258	0	39
p5.4.a	4	1.2	0	0	0	0	0	0
p5.4.b		2.5	0	0	0	0	0	0
p5.4.c		3.8	20	0	20	0	0	8
p5.4.d		5	20	0	20	0	0	8
p5.4.e		6.2	20	0	20	0	0	8
p5.4.f		7.5	80	0	80	0	0	12
p5.4.g		8.8	140	0	140	0	0	16
p5.4.h		10	140	0	140	0	0	16
p5.4.i		11.2	240	0	240	0	0	20
p5.4.j		12.5	340	0	340	0	0	24
p5.4.k		13.8	340	0	340	0	0	24
p5.4.l		15	430	1	430	1	0	26
p5.4.m		16.2	555	0	555	0	0	29
p5.4.n		17.5	620	0	620	0	0	32
p5.4.o		18.8	690	1	690	1	0	34
p5.4.p		20	790	1	765	729	3.16	35
p5.4.q		21.2	860	1	860	1	0	40
p5.4.r		22.5	960	14	960	14	0	44
p5.4.s		23.8	1,055	99	–	–	–	–
p5.4.t		25	1,160	254	1,160	254	0	44
p5.4.u		26.2	1,300	732	1,300	732	0	48
p5.4.v		27.5	1,320	446	1,320	446	0	52
p5.4.w		28.8	1,420	1,946	–	–	–	–

Table 7 TOP – results for instances with 100 vertices

Instance	*m*	*L*	MP		Integer solution			
			Val	CPU(s)	Val	CPU(s)	gap	#
p4.2.a	2	25	206	0	206	0	0	12
p4.2.b		30	341	0	341	0	0	23
p4.2.c		35	458	3	452	6	1.31	30
p4.2.d		40	535.5	60	531	154	0.84	34
p4.2.e		45	623.75	1,984	618	4,823	0.92	39
p4.3.a	3	16.7	0	0	0	0	0	0
p4.3.b		20	38	0	38	0	0	6
p4.3.c		23.3	193	0	193	0	0	15
p4.3.d		26.7	339	1	335	1	1.17	26
p4.3.e		30	468.75	1	468	1	0.16	32
p4.3.f		33.3	584.5	2	579	17	0.94	39
p4.3.g		36.7	656.375	12	653	52	0.51	42
p4.3.h		40	735.375	78	729	801	0.86	49
p4.3.i		43.3	813.625	584	809	4,920	0.56	58
p4.4.a	4	12.5	0	0	0	0	0	0
p4.4.b		15	0	0	0	0	0	0
p4.4.c		17.5	0	0	0	0	0	0
p4.4.d		20	38	0	38	0	0	6
p4.4.e		22.5	183	0	183	0	0	17
p4.4.f		25	324	0	324	0	0	25
p4.4.g		27.5	462	0	461	0	0.21	35
p4.4.h		30	571	2	571	2	0	38
p4.4.i		32.5	665.4	3	657	23	1.26	47
p4.4.j		35	741.472	7	732	141	1.27	51
p4.4.k		37.5	831.945	20	821	558	1.31	59
p4.4.l		40	893.303	64	–	–	–	–

linear relaxation (11 instances). Dashes are then used in the tables to indicate that the optimal integer solution was not found.

In many cases, the upper bound provided by MP is integer and the optimal solution is found at the root node of the Branch & Price tree. As expected, instances are more easily solved for small values of L. Note that, though the tables do not point it out, almost all of the computing time is spent during the subproblem phase.

We evaluate the impact of the different acceleration techniques in Tables 9 and 10. Table 9 provides the average CPU time in seconds required for solving the linear relaxation (column CPU(s)) and the number of times the relaxation is solved with a 2 hour time limit (column #). Table 10 provides the same information for the integer solution.

These tables compare four versions of the Branch & Price algorithm:

- BP is the Branch & Price algorithm with no acceleration technique,
- BP + LDS is the Branch & Price algorithm with the limited discrepancy search component,

Table 8 TOP – results for instances with 102 vertices

Instance	m	L	MP		Integer solution			
			Val	CPU(s)	Val	CPU(s)	gap	#
p7.2.a	2	10	30	0	30	0	0	4
p7.2.b		20	64	0	64	0	0	6
p7.2.c		30	101	0	101	0	0	8
p7.2.d		40	190	0	190	0	0	12
p7.2.e		50	290	1	290	1	0	17
p7.2.f		60	387	55	387	55	0	23
p7.3.a	3	6.7	0	0	0	0	0	0
p7.3.b		13.3	46	0	46	0	0	6
p7.3.c		20	79	0	79	0	0	8
p7.3.d		26.7	117	0	117	0	0	10
p7.3.e		33.3	175	0	175	0	0	13
p7.3.f		40	247	1	247	1	0	17
p7.3.g		46.7	344	0	344	0	0	21
p7.3.h		53.3	429	3	425	8	0.93	27
p7.3.i		60	496.976	132	487	3,407	2.00	31
p7.3.j		66.7	570.5	2,654	–	–	–	–
p7.4.a	4	5	0	0	0	0	0	0
p7.4.b		10	30	0	30	0	0	4
p7.4.c		15	46	0	46	0	0	6
p7.4.d		20	79	0	79	0	0	9
p7.4.e		25	123	0	123	0	0	11
p7.4.f		30	164	0	164	0	0	15
p7.4.g		35	217	0	217	0	0	18
p7.4.h		40	285	0	285	0	0	21
p7.4.i		45	366	0	366	0	0	25
p7.4.j		50	462	1	462	1	0	31
p7.4.k		55	524.607	6	520	73	0.87	34
p7.4.l		60	593.625	57	590	778	0.61	40
p7.4.m		65	660.667	730	–	–	–	–

Table 9 TOP – impact of the acceleration techniques for the relaxed solution

n	BP		BP + LDS		BP + LLME		BP + LDS + LLME	
	CPU(s)	#	CPU(s)	#	CPU(s)	#	CPU(s)	#
21	0.2	**33**	0.06	**33**	0.09	**33**	0.03	**33**
32	232.9	**52**	91.6	**52**	117.2	51	92.7	**52**
33	170.14	49	168.16	**50**	217.28	**50**	184.6	**50**
64	82.1	36	257.16	**37**	122.5	36	51.3	**37**
66	315.6	47	285.9	**50**	332.5	47	224.48	**50**
100	134.6	**26**	220.5	**26**	107.15	**26**	108.5	**26**
102	60	27	221.5	**29**	298.3	28	125.5	**29**

Table 10 TOP – impact of the acceleration techniques for the integer solution

n	BP		BP + LDS		BP + LLME		BP + LDS + LLME	
	CPU(s)	#	CPU(s)	#	CPU(s)	#	CPU(s)	#
21	0.15	**33**	0.06	**33**	0.09	**33**	0.06	**33**
32	233.9	**52**	92.7	**52**	118	51	38.23	51
33	220.7	48	110.9	47	134.29	48	103.84	**50**
64	222.3	35	251	35	308.17	35	286.1	**36**
66	330.28	46	207	**48**	348.02	46	200.64	**48**
100	435.4	**25**	599.12	**25**	373.96	**25**	459.9	**25**
102	295.9	**27**	211.3	**27**	293.3	**27**	203.2	**27**

– BP + LLME is the Branch & Price algorithm with the label loading and the meta extensions components,
– BP + LDS + LLME is the Branch & Price algorithm with all acceleration techniques.

Values in bold highlight the method solving the maximal number of instances. CPU times have to be considered more carefully, as the mean time depends on the set of instances solved.

These tables show that the algorithm is able to solve a great deal of instances without resorting to the acceleration techniques. Actually, many instances are solved at the root node of the search tree with a very small computing time.

Table 11 Profit values for Gueguen (1999) data sets

#	Profit	#	Profit	#	Profit	#	Profit	#	Profit
1	0	21	49	41	43	61	18	81	29
2	28	22	22	42	36	62	4	82	19
3	9	23	5	43	47	63	33	83	36
4	40	24	0	44	46	64	2	84	30
5	29	25	0	45	26	65	0	85	28
6	23	26	18	46	7	66	45	86	18
7	17	27	26	47	23	67	13	87	7
8	44	28	28	48	11	68	13	88	11
9	41	29	30	49	43	69	29	89	21
10	37	30	30	50	10	70	34	90	40
11	8	31	8	51	38	71	41	91	25
12	42	32	33	52	42	72	36	92	49
13	35	33	22	53	49	73	24	93	37
14	25	34	17	54	49	74	10	94	17
15	15	35	2	55	30	75	37	95	8
16	0	36	30	56	19	76	23	96	32
17	4	37	39	57	13	77	22	97	24
18	18	38	40	58	14	78	47	98	3
19	7	39	25	59	42	79	37	99	34
20	8	40	15	60	1	80	5	100	25

However, the impact of the acceleration techniques is often important for difficult instances. The LDS component behaves better than the label loading–meta extensions combination most of the times. But these two techniques combine well and the version of the algorithm implementing all the acceleration techniques clearly outperforms the three others.

4 Application to the selective vehicle routing problem with time windows

The SVRPTW both generalizes the TOP and the vehicle routing with time windows (VRPTW). Some new constraints have to be added to the generic description of Sect. 2.1. Each customer is defined with a demand, a service time and a time window for the visit. Vehicles have a given capacity. Vehicles routes thus have to respect simultaneously the length limitation L, a capacity constraint

Table 12 Results for the problem r101 with 50 customers

L	m	Gueguen				Our algorithm			
		rel	CPU(s)	int	CPU(s)	rel	CPU(s)	int	CPU(s)
50	1	123	0.2	123	0.26	123	0.05	123	0.05
	2	215	0.29	215	0.35	215	0.12	215	0.12
	3	287	0.48	282	2.52	287	0.13	282	0.28
	4	334	0.38	329	2.74	334	0.15	329	0.34
	5	381	0.39	376	3.75	374	0.15	369	0.4
	6	427	0.38	422	3.01	412	0.13	407	0.51
	7	473	0.39	468	2.57	443.5	0.11	439	0.28
	8	517	0.39	512	1.58	456	0.11	456	0.14
	9	560	0.48	555	1.79	456	0.07	456	0.13
	10	603	0.39	598	1.58	456	0.05	456	0.09
100	1	202	0.38	202	0.44	202	0.38	202	0.38
	2	374	0.99	374	1.05	374	0.47	374	0.47
	3	520	1.02	520	1.08	520	0.59	520	0.59
	4	651	1.54	651	1.6	651	1.26	651	1.26
	5	761.5	1.8	761	4.39	761.5	1.74	761	2.96
	6	857.2	2.3	852	22.07	857.2	1.79	852	9.49
	7	973.25	2.07	935	23.39	937.25	2.61	935	6.96
	8	1009.13	2.44	1,006	28.18	1009.12	2.45	1,006	8.77
	9	1075.67	2.39	1,071	9.88	1075.67	2.47	1,071	4.93
	10	1127.37	2.99	1,126	5.82	1127.38	3.08	1,125	7.34
150	1	210	0.61	210	0.66	210	0.45	210	0.45
	2	383	0.83	383	0.89	383	0.57	383	0.57
	3	542	1.7	542	1.75	542	0.66	542	0.66
	4	695	1.37	695	1.43	695	1.03	695	1.03
	5	819	1.97	819	2.03	819	2.05	819	2.05
	6	917	2.04	917	2.1	917	2.73	917	2.73
	7	1,003	1.97	1,003	2.03	1,003	3.91	1,003	3.91
	8	1,068	2	1,068	2.05	1,068	3.41	1,068	3.41
	9	1,120	2.82	1,116	785.9	1,120	2.46	1,116	7.72
	10	1,148	2.82	1,146	1098.46	1,148	2.93	1,146	6.4

Table 13 Results for the problem r101 with 100 customers

L	m	Gueguen				Our algorithm			
		rel	CPU(s)	int	CPU(s)	rel	CPU(s)	int	CPU(s)
50	1	166	0.82	166	0.88	164	0.67	164	0.67
	2	324	1.64	324	1.7	313	0.85	313	0.85
	3	454.5	2.11	454	4.88	445	1.32	445	1.32
	4	583.5	2.11	583	5.22	574	1.28	574	1.28
	5	702.5	2.54	702	6.06	693	1.23	693	1.23
	6	782.8	3.61	776	21.00	772.25	2.55	767	7.04
	7	861.8	3.38	860	13.97	851.25	2.07	851	3.84
	8	934	4.02	934	4.08	925	1.97	925	1.97
	9	995	4.89	995	4.95	995	1.97	995	1.97
	10	1,042.82	5.23	1,042	8.92	1,042	3.05	1,042	3.05
100	1	280	12.22	280	12.28	280	24.37	280	24.37
	2	548	22.76	548	22.82	541	34.93	541	34.93
	3	777	36.47	777	36.53	770	80.42	770	80.42
	4	991	46.44	991	146.5	984	53.47	983	96.45
	5	1,201	48.19	1,201	48.25	1,190.2	65.22	1,190	103.53
	6	1,377.5	59.95	1,374	62.94	1,372	117.8	1,372	117.8
	7	1,541.25	70.04	1,540	91.96	1,538	102.58	1,538	102.58
	8	1,685.88	85.41	1,677	3,231.03	1,681.2	54.65	1,675	424.33
	9	1,822.14	105.94	1,818	1,351.58	1,814	74.92	1,814	74.92
	10	1,932.53	108.45	–	–	1,930.29	112.35	–	–
150	1	320	15.48	320	15.54	320	15.85	320	15.85
	2	590	35.43	590	35.49	590	17.89	590	17.89
	3	820	62.8	820	62.86	820	41.2	820	41.2
	4	1049	86.13	1049	86.19	1049	52.49	1049	52.49
	5	1260	92.15	1,255	160.67	1,260	52.46	1,255	245.72
	6	1,445.83	117.71	1,444	732.86	1,445.5	61.8	1,444	148.28
	7	1,613	110.00	–	–	1,611.5	79.02	1,606	303.9
	8	1,761.25	136.84	–	–	1,758	73.35	1,752	211.88
	9	1,899.75	157.3	–	–	1,894	74.58	1,894	74.58
	10	2,022.67	191.22	–	–	2,019	62.22	2,012	798.43

and time windows for visited customers. Routes start and end in a unique depot. Viewing the problem as an extension of the VRPTW, it corresponds to the case where all customers cannot be visited for some logistic reason.

The new constraints only influence the solution scheme at the subproblem level. Two new resources are introduced to take account of time and load. These two resources are managed exactly as in the VRPTW situation (see, e.g., Desrochers et al. 1992): the value of the load resource increases after each customer visit and is not allowed to exceed the capacity of the vehicle; the value of the time resource increases according to arc travel times, service times and waiting times imposed by earliest arrival times on nodes, and is not allowed to exceed latest arrival times.

Computational experiments have been conducted on Gueguen (1999) data sets. These data sets extend Solomon's data sets defined for the VRPTW. A value for L and randomly generated profits are simply added. Following Gueguen (1999), we use a unique set of profit values, described in Gueguen (1999) and that we recall in Table 11. We use a PC Pentium II 333 MHz, with a time limit of 1 h, as in Gueguen (1999).

Gueguen (1999) describes results on instance r101 with 50 and 100 customers, for different values of m and L. We evaluate our algorithm on the same instances. Results are given in Tables 12 and 13, where:

- rel is the linear relaxation value,
- CPU(s) is the CPU time in seconds of the resolution,
- int is the integer solution value.

Note that optimal solutions found by both algorithms are different in many cases, which implies some malfunctioning. Gueguen (2006) indicates that the preliminary results from Gueguen (1999) were not entirely reliable: for some problems, the initial sets of columns contained some infeasible columns, which potentially improved the value of optimal solutions. In these conditions it is rather difficult to compare both algorithms. However, we can draw the same conclusions as Gueguen (1999) and as for the TOP: the gap between the linear relaxation and the integer solution value is very small, and L has a deep impact on the computing ability of the algorithm.

5 Conclusion

In this article, we present a Branch & Price algorithm for the optimal resolution of orienteering problems. As far as we know, this is the first exact algorithm capable of solving efficiently different variants of orienteering problems. Our Branch & Price algorithm includes branching rules specifically devoted to orienteering problems and adapt acceleration techniques to this context.

Computational experiments demonstrate the ability of our algorithm for solving instances of medium size. Another interesting contribution of our work is to provide a comparison tool for evaluating heuristic procedures, especially for the TOP for which such a tool is drastically missing.

Acknowledgements The authors wish to thank two anonymous referees for their very helpful comments and suggestions.

References

Archetti C, Hertz A, Speranza MG (2005) Metaheuristics for the team orienteering problem. Technical Report GERAD-2005-47, Groupes d'Etudes et de Recherche en Analyse des Dcisions

Butt SE, Ryan DM (1999) An optimal solution procedure for the multiple tour maximum collection problem using column generation. Comput Oper Res 26(4):427–441

Chao I-M, Golden BL, Wasil EA (1996) The team orienteering problem. Eur J Oper Res 88(3): 464–474

Desaulniers G, Desrosiers J, Solomon MM (eds) (2005) Column generation. GERAD 25th Anniversary Series. Springer, Berlin Heidelberg New York

Desrochers M, Desrosiers J, Solomon MM (1992) A new optimization algorithm for the vehicle routing problem with time windows. Oper Res 40(2): 342–354

Feillet D, Dejax P, Gendreau M, Gueguen C (2004) An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle routing problems. Networks 44 (3): 216–229

Feillet D, Dejax P, Gendreau M(2005a) Traveling salesman problems with profits. Transp Sci 39(2): 188–205

Feillet D, Gendreau M, Rousseau LM (2005b) New refinements for the solution of vehicle routing problems with branch and price. Technical Report CRT-2005-08, Centre de Recherche sur les Transports

Gueguen C (1999) Méthodes de résolution exacte pour les problèmes de tournées de véhicules. Thése de doctorat, école Centrale Paris

Gueguen C (2006) Private communication

Harvey W, Ginsberg M (1995) Limited discrepancy search. In: Proceedings of the 14th international joint conference on artificial intelligence (IJCAI-95), Morgan Kaufmann, Montréal, pp. 607–615

Hayari N, Manier M-A, Bloch C, El Moudni A (2003) Un algorithme évolutionniste pour le problème de tournées sélectives avec contraintes de fenêtre de temps. In 4ème Conférence Francophone de MOdélisation et SIMulation MOSIM'03, Toulouse

Irnich S, Desaulniers G (2005) Shortest path problems with resource constraints. In: Desaulniers G, Desrosiers J, Solomon MM, (eds) Column generation, GERAD 25th Anniversary Series, chap 2, pp. 33–65, Springer, Berlin Heidelberg New York

Laporte G, Martello S (1990) The selective traveling salesman problem. Discrete Appl Math 26:193–207

Tang H, Miller-Hooks E (2005) A tabu search heuristic for the team orienteering problem. Comput Oper Res 32(6): 1379–1407