International Journal of Engineering, Mathematical and Physical Sciences
ISSN: 2517-9934
Vol:8, No:9, 2014

A Hybrid Heuristic for the Team Orienteering Problem

Adel Bouchakhchoukha, Hakim Akeb

Abstract—In this work, we propose a hybrid heuristic in order to solve the Team Orienteering Problem (TOP). Given a set of points (or customers), each with associated score (profit or benefit), and a team that has a fixed number of members, the problem to solve is to visit a subset of points in order to maximize the total collected score. Each member performs a tour starting at the start point, visiting distinct customers and the tour terminates at the arrival point. In addition, each point is visited at most once, and the total time in each tour cannot be greater than a given value. The proposed heuristic combines beam search and a local optimization strategy. The algorithm was tested on several sets of instances and encouraging results were obtained.

Keywords—Team Orienteering Problem, Vehicle Routing, Beam Search, Local Search.

I. Introduction

Vehicle Routing Problems (VRP) are very well studied in the literature because they have many practical applications in Industry and Logistics. In a standard VRP, we have to visit a given set of points (vertices) called *customers* by using one or more vehicle(s). The objective is to optimize one or more objective(s).

The most known vehicle routing problem is of course the Traveling Salesman Problem (TSP) [10] that consists to visit a set of customers (each one must be visited exactly once) by using one vehicle. The objective in TSP is to minimize the total distance traveled. This is equivalent to compute a Hamiltonian circuit in a complete graph (the shortest path that visits each node of the graph exactly once). In some circumstances, the customers have to be visited during a given period of time (*time window*), the corresponding problem is called “Vehicle Routing Problem with Time Windows” (VRPTW) [7]. In practical cases, several vehicles are used in order to serve all the customers and the objectives to optimize are the number of vehicles (to minimize) and the total distance that has also to be minimized, so this corresponds to a multi-objective problem.

In some cases, a score (or *benefit*) is assigned to each customer or vertex and only a subset of customers are visited because the length and/or time are limited. The objective is to maximize the collected scores. This situation corresponds to a category of problems known as the *Orienteering Problem* (OP) [12]. This can be seen as a combination of the TSP and the Knapsack Problem (KP).

In this paper we study the *Team Orienteering Problem* (TOP) defined by Chao et al. [5], also known as *Multiple Tour Maximum Collection Problem* (MTMCP). Here we have to determine several paths that maximize the collected scores (or benefits), each path is bounded by a length or time. TOP is then equivalent to an OP executed by a *team*, i.e., several members or vehicles, each member has to perform a path (or tour) of maximum collected score but not exceeding the maximum length or time. So in TOP, we have:

- A set $V = \{v_0, ..., v_{n+1}\}$ of vertices or points to visit. Vertex v_0 corresponds to the starting point while v_{n+1} is the end (arrival) point. No score is associated with these two vertices.
- A non-negative score S_i is associated with each vertex $v_i \in V$, the score associated with v_0 and v_{n+1} is equal to 0.
- The time (or length) t_{ij} needed to go from vertex i to vertex j is known.
- m paths $\mathbb{P} = \{P_1, ..., P_m\}$ have to be performed by m members (vehicles for example). Each path begins at the starting point v_0, visits a distinct set of points, and terminates at the end point v_{n+1}. Each visited vertex in V belongs then to one and only one path.
- The time needed to perform each path $P_k, 1 \leq k \leq m$, denoted by $T(P_k)$ must not exceed a fixed value T_{max} that is the same for all paths. This constraint is indicated in (1).

$$T(P_k) \leq T_{max},\ \forall P_k \in \mathbb{P} \tag{1}$$

The objective is then to maximize the sum of the collected scores in $\mathbb{P}$, the set of all paths, each path $P_k \in \mathbb{P}$ has a total time (or length) not exceeding T_{max}. Note that if the speed is equal to one unit, then the time and the length are equivalent. The objective associated with the collected score is indicated in (2).

$$\max \quad S(\mathbb{P}) = \sum_{k=1}^{m} S(P_k) \tag{2}$$

Fig. 1 shows an example of a graph containing 13 points ($|V| = 13$). The values indicate the score associated to each vertex. The start point corresponds to the triangle ▲ while the end (or final) point is represented by a square ■. There are two paths to compute ($|\mathbb{P}| = m = 2$). The first path P_1 is indicated in the right side of the graph and the corresponding collected score is $S(P_1) = 2 + 8 + 4 + 1 = 15$. Path P_2 has a score $S(P_2) = 2 + 7 + 8 + 3 = 20$. The collected score in the solution is then $S(\mathbb{P}) = 15 + 20 = 35$.

A. Bouchakhchoukha is a PhD student at MSE Université Paris 1, Panthéon Sorbonne, 106-112 Bd de l’Hôpital, 75013 Paris, France. (e-mail: adel.bouchakhchoukha@malix.univ-paris1.fr).

H. Akeb is a Professor at ISC Paris Business School, 22 Bd du Fort de Vaux, 75017 Paris, France. (e-mail: hakeb@iscparis.com).

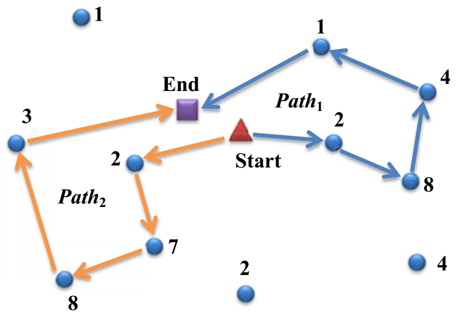

Fig. 1 An example of solution for the Team Orienteering Problem with two paths

II. Literature Review

To our knowledge, the Team Orienteering Problem was first studied by Butt and Cavalier [4] under the name of *Multiple Tour Maximum Collection Problem* (MTMCP). The authors proposed a heuristic named MAXIMP (Maximize Importance) that is composed of four steps. The main idea of MAXIMP is to assign weights to node pairs. This then defines a priority list denoted by WGT that indicates the order in which the pairs of vertices are taken in order to construct the m tours. The term of "TOP" was used for the first time by Chao et al. [5]. The authors described a problem where a team is composed of m members, each member will perform a path by starting at the start point, going through a subset of points, and terminating at the end point without exceeding a maximum time. A score is associated with each point and the goal is to maximize the collected (total) score. The authors presented a heuristic in order to solve a large variety of problems that were generated by the authors. The instances contain from 21 to 102 points while the number of paths varies from 2 to 4. The heuristic developed uses different tools including exchange of two points between two paths and moving a point from a path to another one in order to increase the collected score in the solution. In addition, Local Search (2-opt algorithm) is used in order to decrease the time in a path, this may allow including new points and then increasing the collected score.

Several (meta-)heuristics were used in the literature in order to solve the TOP. Bouly et al. [1] proposed a memetic algorithm that is based on a hybrid genetic algorithm. Lin [11] developed a simulated annealing based algorithm. Kim et al. [9] proposed an augmented large neighborhood search (LNS) for the same problem. Finally, Hu and Lim [8] proposed an iterative three-component heuristic for a TOP with time windows, i.e., a time window is associated with each vertex. The reader can refer to Vidal et al. [13] that published a survey on the different heuristics used for solving the multi-attribute vehicle routing problems.

There also exists exact methods that solves the team orienteering problem. For example, Butt and Ryan [3] used an exact method based on Column Generation in order to solve the Multiple Tour Maximum Collection Problem (MTMCP). The method, denoted by MAXREW, is able to obtain optimal solutions, even when the number of nodes is large (100 nodes), but with a greater computation time. Boussier et al. [2] proposed an exact algorithm for the TOP based on Branch-and-Price where the authors presented also some techniques that accelerate the search and instances with up to 100 customers are solved.

In this paper we propose a hybrid heuristic for the TOP. The heuristic combines beam search and a local optimization based on the 3-opt strategy that works similarly as the well-known 2-opt method [6]. In addition, the proposed heuristic implements a pre-processing step that serves to eliminate vertices that will never belong to a feasible solution.

III. A Heuristic for the Team Orienteering Problem

In the TOP some vertices can be eliminated because they can never belong to a feasible solution because they violate the time constraint. After eliminating these vertices, we obtain set V' defined as follows:

$$V' = \{v_i \in V\} \mid (t_{v_0,v_i} + t_{v_i,v_{n+1}}) \leq T_{max} \qquad (3)$$

Equation (3) defines then the set of vertices to take into account in order to compute a solution for the TOP. More precisely, if the start and end point are distinct, then vertices in V' belong to an ellipse where the foci are the start point v_0 and the end point v_{n+1} and the length of the major axis is equal to T_{max}. Chao et al. [5] called this ellipse the "T_{max} ellipse". When the start point is the same than the end point, then we obtain a circle of diameter T_{max}.

Below will be described the proposed algorithm that uses two techniques:

- Beam search that serves to compute paths of maximum score.
- A local optimization based on the 3-opt algorithm in order to decrease the total time in the current partial solution. This procedure is called at each step of beam search.

A. Beam Search for Computing Paths

Beam search is a tree search that computes several paths in parallel, and the best feasible one (with the highest score) is chosen as the final solution. This is actually an optimization of the *Best First Search* since it selects, at each level of the tree, the best ω nodes, where ω is an integer value called the *beam width.*

The adaptation of Beam Search to our problem is given in Algorithm 1 (BSCBP) that receives as input parameter the set of vertices V' that belong to the T_{max} ellipse. The output of BSCBP is the set of best paths found $\mathbb{P} = \{P_1, ..., P_m\}$ that maximize the collected score.

The nodes of the current level in the tree are stored in a list denoted by B (line 1) in the algorithm, and the offspring nodes (created when branching from the nodes in B) are stored in list B_{off}. Remember that path $P_k \in \mathbb{P}$ must start from the start point v_0, visits a subset of vertices in V' and ends at vertex v_{n+1}.

Input: Set $V' \subseteq V$ of vertices that belong to the T_{max} ellipse.
Output: The best paths found $\mathbb{P} = \{P_1, ..., P_m\}$ that maximize the collected score.

1: Let B be the set containing the nodes at a given level of the tree;
2: Let B_{off} be the offspring nodes (descendants of nodes in B);
3: **for** $k = 1$ to m **do**
4: $\quad \eta_0 \leftarrow \{P^+ = \{v_0\}, P^- = V'\}$ (the root node)
5: $\quad$ Set $\eta_0.\mathrm{score} \leftarrow 0$ and $\eta_0.\mathrm{time} \leftarrow 0$;
6: $\quad$ Set $B \leftarrow \{\eta_0\}$ and $\ell \leftarrow 0$;
7: $\quad \eta^* \leftarrow \eta_0$; (the best solution found for the k^{th} path)
8: $\quad$ **while** $(B \neq \emptyset)$ **do**
9: $\quad\quad$ Branch out of each node $\eta_{\ell_j} \in B$ and create the offspring nodes B_{off};
10: $\quad\quad \ell \leftarrow \ell + 1$;
11: $\quad\quad$ **for each** nodes $\eta_{\ell_j} \in B_{off}$ **do**
12: $\quad\quad\quad$ Apply the 3-opt local optimization on the partial path P_j^+ of the node in order to try to decrease $\eta_{\ell_j}.\mathrm{time}$;
13: $\quad\quad$ **end for**
14: $\quad\quad$ Remove from B_{off} the nodes that will violate the T_{max} constraint if adding the end point v_{n+1};
15: $\quad\quad$ **if** $P^- = \emptyset$ for a node $\eta_{\ell_j} \in B_{off}$ **then**
16: $\quad\quad\quad$ Add vertex v_{n+1} (end point) to that path and compute the total time and score;
17: $\quad\quad\quad$ **if** (η_{ℓ_j}.score$>$ η^*.score) **then**
18: $\quad\quad\quad\quad \eta^* \leftarrow \eta_{\ell_j}$;
19: $\quad\quad\quad\quad$ Remove η_{ℓ_j} from B_{off};
20: $\quad\quad\quad$ **end if**
21: $\quad\quad$ **end if**
22: $\quad\quad$ Sort nodes in B_{off} according to the selection criterion ρ and keep only the $\max(\omega, |B_{off}|)$ first nodes, remove the other nodes from B_{off};
23: $\quad\quad B \leftarrow B_{off}$;
24: $\quad\quad B_{off} \leftarrow \emptyset$;
25: $\quad$ **end while**
26: $\quad$ Assign to $P_k \in \mathbb{P}$ the path P^+ stored in node η^*;
27: $\quad$ Update set V' by removing the vertices used in path P_k;
28: **end for**

Algorithm 1: BSCBP (Beam Search for Computing the Best Paths).

The m paths are computed sequentially by the **for** loop (lines 3 – 28). For each path, instructions between lines 4 and 27 are executed. A node η_ℓ at level ℓ in the tree contains the following elements: the path under construction P^+, the set of vertices P^- that are not yet visited, the total time T corresponding to path P^+, and finally the total collected score S in P^+. These two last parameters are designed by $\eta_\ell.\mathrm{score}$ and $\eta_\ell.\mathrm{time}$ respectively.

The root node of the tree η_0 is initialized at line 4 where $P^+ = \{v_0\}$ and $P^- = V'$. At line 5, the score as well as the total time are set to 0. List B is then initialized with η_0 and the current level ℓ is set to 0 (line 6). The best node η^*, that contains the best solution found so far, is set to the root node (line 7).

Computing paths is done inside the **while** loop that begins at line 8. Indeed, at each level ℓ of the tree, list B contains the nodes corresponding to the partial paths. Branching from node $\eta_{\ell_j} = \{P_j^+, P_j^-\} \in B$ means that the next vertex to visit will be chosen from set P_j^-. So the node can generate at most $|P_j^-|$ descendants and then the current level, that contains $|B|$ nodes, can generate at most $\sum_{j=1}^{|B|}(|P_j^-|)$ descendants. These offspring nodes are then stored in list B_{off} (line 9). After that, the level is incremented at line 10. At line 11, a local optimization, that is based on 3-opt, tries to rearrange the arcs in each path in order to decrease the total time. The goal is to be able after that to insert more vertices in a path in order to increase the collected score (the 3-opt optimization method is given in Algorithm 2). Then, the nodes containing non-feasible paths are removed from B_{off} (line 14).

After that, if set P^- is empty (line 15) in a given node $\eta_{\ell_j} \in B_{off}$, then we add the last arc by connecting the last vertex in P^+ with the end point v_{n+1}. The obtained score is then compared to the best known one and the best node η^* is updated if a greater score is obtained (lines 18) and the node is then removed from B_{off} (line 19).

The next step consists to filter list B_{off} in order to keep only the ω best ones. This is done by sorting the nodes in B_{off}, by using a given criterion ρ, from the most important to the least important one, and then keeping the first (best) ones, the other nodes are removed from B_{off}. The computational investigation shows that the best criterion to sort the nodes is that which chooses the nearest vertex as the next one to visit. If there are more than one such vertices then the vertex with the greatest score is chosen. The remaining nodes in B_{off} are then assigned to B (line 23) and B_{off} reset to the empty set (line 24).

The **while** loop stops when no branching is possible, i.e., when B becomes empty. Then the current path P_k is assigned with the path P^+ computed in the best node η^* (line 26). The last instruction (line 27) consists to update the set of vertices V' by removing from it the vertices used in the last computed path P_k.

So the output of algorithm BSCBP is the set $\mathbb{P}$ containing the m best paths found. The total score corresponds to the sum of scores in each path $P_k \in \mathbb{P}, k = 1, ..., m$.

B. Local Search for Decreasing the Time in a Path

In order to try to decrease the total time in a path, a local optimization is used. The method is based on the so-known 3-opt strategy given in Algorithm 2. The idea is to take three non-successive arcs $(v_i \to v_{i+1})$, $(v_j \to v_{j+1})$, and $(v_k \to v_{k+1})$, and replace them by arcs $(v_i \to v_j)$, $(v_{i+1} \to v_k)$, and $(v_{j+1} \to v_{k+1})$ if the obtained total time in the path decreases. This process is repeated until there is no improvement after trying all the combinations. Note that after changing arcs, the direction of some other arcs must be inverted.

Fig. 2 gives an example where arcs $(A \to B)$, $(C \to D)$, and $(E \to F)$ are replaced by arcs $(A \to C)$, $(B \to E)$, and

International Journal of Engineering, Mathematical and Physical Sciences
ISSN: 2517-9934
Vol:8, No:9, 2014

Input: A path P of total time $T(P)$
Output: A path P' with total time $T(P') \leq T(P)$

Improvement $\leftarrow$ true;
while (improvement = true) **do**
  Improvement $\leftarrow$ false;
  for each vertex $v_i \in P$ **do**
    for each vertex $v_j \in P$ $(j \neq i-1, j \neq i+1)$ **do**
      for each vertex $v_k \in P$ $(k \neq j-1, k \neq j+1, k \neq i+1, k \neq i-1)$ **do**
        if $(t_{v_i,v_{i+1}} + t_{v_j,v_{j+1}} + t_{v_k,v_{k+1}}) > (t_{v_i,v_j} + t_{v_{i+1},v_k} + t_{v_{j+1},v_{k+1}})$ **then**
          Replace arcs $(v_i \rightarrow v_{i+1}), (v_j \rightarrow v_{j+1})$ and $(v_k \rightarrow v_{k+1})$ by $(v_i \rightarrow v_j), (v_{i+1} \rightarrow v_k)$ and $(v_{j+1} \rightarrow v_{k+1})$ respectively;
          Improvement $\leftarrow$ true;
        end if
      end for
    end for
  end for
end while

Algorithm 2: The 3-opt algorithm for decreasing the total time in a path.

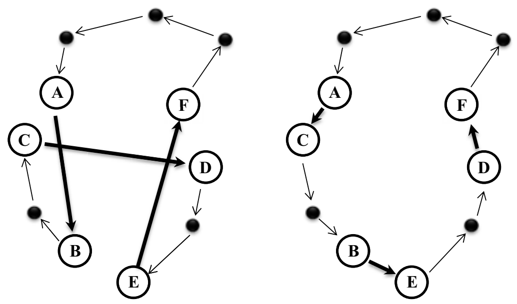

Fig. 2 An example of changing of three arcs in a path

$(D \rightarrow F)$. Note that some arcs are inverted in order to keep a circuit in the path.

IV. Computational Results

The proposed algorithm is implemented using the C++ language and executed under Windows 7 environment on a computer that has 2 GB of RAM and a 2.26 GHz processor.

Three sets of instances were considered, namely p1, p2, and p3, and proposed by Chao et al. [5]. The number V of vertices in these problem sets are 21, 32 and 33 respectively. More precisely, the name of each instance in each set is in the form “p[number1].[number2].[letter]” where:

- [number1] represents the group number (1, 2, or 3) in our case.
- [number2] is the number m of members with $2 \leq m \leq 4$.
- [letter] is an alphabetical letter that serves to identify the T_{max} value.

TABLE I
Results Obtained on the 54 Instances of the First Set (p1): $|V| = 21$ Vertices

Instance	Best Lit.	BSCBP	CPU time (sec)
p1.2.a	0	**0**	0
p1.2.b	15	**15**	< 1
p1.2.c	20	**20**	< 1
p1.2.d	30	**30**	< 1
p1.2.e	45	**45**	< 1
p1.2.f	80	**80**	3
p1.2.g	90	**90**	181
p1.2.h	110	**110**	1006
p1.2.i	135	**135**	1279
p1.2.j	155	**155**	1288
p1.2.k	175	**175**	1515
p1.2.l	195	**195**	1609
p1.2.m	215	**215**	1741
p1.2.n	235	**235**	2055
p1.2.o	240	**240**	2398
p1.2.p	250	**250**	5707
p1.2.q	265	**265**	5611
p1.2.r	280	**280**	7102
p1.3.a	0	**0**	0
p1.3.b	0	**0**	0
p1.3.c	15	**15**	< 1
p1.3.d	15	**15**	< 1
p1.3.e	30	**30**	< 1
p1.3.f	40	35	< 1
p1.3.g	50	**50**	1
p1.3.h	70	**70**	17
p1.3.i	105	100	171
p1.3.j	115	110	411
p1.3.k	135	**135**	746
p1.3.l	155	150	1251
p1.3.m	175	**175**	1366
p1.3.n	190	**190**	1998
p1.3.o	205	**205**	1820
p1.3.p	220	**220**	1817
p1.3.q	230	225	1808
p1.3.r	250	215	1955
p1.4.a	0	**0**	0
p1.4.b	0	**0**	0
p1.4.c	0	**0**	0
p1.4.d	15	**15**	< 1
p1.4.e	15	**15**	< 1
p1.4.f	25	**25**	< 1
p1.4.g	35	**35**	< 1
p1.4.h	45	**45**	< 1
p1.4.i	60	**60**	< 1
p1.4.j	75	**75**	2
p1.4.k	100	**100**	7
p1.4.l	120	**120**	25
p1.4.m	130	**130**	370
p1.4.n	155	**155**	533
p1.4.o	165	**165**	791
p1.4.p	175	**175**	929
p1.4.q	190	**190**	1175
p1.4.r	210	**210**	1249
Average	112.04	110.93	1248.43

So in fact, for the same problem that have the same vertices with the same coordinates and associated scores, parameters [number2] and [letter] serve to create several versions by modifying value of m and the value of T_{max}. For example for problem “p1.3.a”, there are 32 vertices including the start and end points, $m = 3$ and $T_{max} = 1.7$. and in problem “p1.3.k” we have exactly the same vertices (with the same scores) but the value of T_{max} becomes 18.3. Then, in this second version, the paths become longer and the collected score higher.

The coordinates (x_i, y_i) of the vertices $v_i \in V$ are indicated

TABLE II
RESULTS OBTAINED ON THE 33 INSTANCES OF THE SECOND SET (P2): $|V| = 32$ VERTICES

Instance	Best Lit.	BSCBP	CPU time (sec)
p2.2.a	90	**90**	< 1
p2.2.b	120	**120**	1
p2.2.c	140	**140**	1
p2.2.d	160	**160**	2
p2.2.e	190	**190**	2
p2.2.f	200	**200**	6
p2.2.g	200	**200**	4
p2.2.h	230	**230**	3
p2.2.i	230	**230**	4
p2.2.j	260	**260**	6
p2.2.k	275	**275**	9
p2.3.a	70	**70**	< 1
p2.3.b	70	**70**	< 1
p2.3.c	105	**105**	< 1
p2.3.d	105	**105**	1
p2.3.e	120	**120**	1
p2.3.f	120	**120**	1
p2.3.g	145	**145**	1
p2.3.h	165	**165**	2
p2.3.i	200	**200**	2
p2.3.j	200	**200**	2
p2.3.k	200	**200**	1
p2.4.a	10	**10**	< 1
p2.4.b	70	**70**	< 1
p2.4.c	70	**70**	< 1
p2.4.d	70	**70**	< 1
p2.4.e	70	**70**	< 1
p2.4.f	105	**105**	< 1
p2.4.g	105	**105**	< 1
p2.4.h	120	**120**	1
p2.4.i	120	**120**	1
p2.4.j	120	**120**	1
p2.4.k	180	**180**	1
Average	140.45	140.45	2.41

in a Euclidean plan and the time that corresponds to arc $(v_i \rightarrow v_j)$ denoted by t_{v_i,v_j} is exactly the euclidean distance between these two vertices, i.e., $t_{v_i,v_j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. So the speed of the member is considered to be one unit.

Tables I–III indicate the results obtained on the three sets of problem instances p1, p2, and p3 respectively. The first column in each table contains the name of the instance, the second column "Best Lit." indicates the best known solution (score) for each instance extracted from [11]. The next column "BSCBP" shows the results obtained by the proposed algorithm, the values are in bold characters when the best known value in the literature is reached. Finally column 4 gives the computation time in seconds.

It is to note that beam search is run for each value of ω (the beam width) in the interval 1 to 200. So the CPU time indicates here the cumulated time.

For the first group (p1) shown in Table I, 48 best known values out of 54 (89%) are reached by algorithm BSCBP. The computation time varies from 0 seconds to 7100 seconds. The time increases when T_{max} increases, this is because longer paths have to be computed, and the 3-opt strategy (that is called in each step of beam search) is time consuming when there are more arcs to process. The last row of Table I indicates the average score obtained on the 54 instances. Its is equal to 110.93 for algorithm BSCBP and 112.04 for the best known results, this correspond to a gap of 0.99%. The last row gives the average computation time, that is equal to 1248 seconds (20.8 minutes).

TABLE III
RESULTS OBTAINED ON THE 60 INSTANCES OF THE THIRD SET (P3): $|V| = 33$ VERTICES

Instance	Best Lit.	BSCBP	CPU time (sec)
p3.2.a	90	**90**	< 1
p3.2.b	150	**150**	2
p3.2.c	180	**180**	9
p3.2.d	220	**220**	129
p3.2.e	260	**260**	25
p3.2.f	300	**300**	1882
p3.2.g	360	**360**	3161
p3.2.h	410	390	4309
p3.2.i	460	450	36541
p3.2.j	510	500	4034
p3.2.k	550	**550**	4591
p3.2.l	590	**590**	5260
p3.2.m	620	**620**	5719
p3.2.n	660	**660**	6128
p3.2.o	690	670	47753
p3.2.p	720	**720**	7044
p3.2.q	760	750	8019
p3.2.r	790	**790**	8570
p3.2.s	800	**800**	8951
p3.2.t	800	**800**	10109
p3.3.a	30	**30**	< 1
p3.3.b	90	**90**	< 1
p3.3.c	120	**120**	1
p3.3.d	170	**170**	26
p3.3.e	200	**200**	6
p3.3.f	230	**230**	12
p3.3.g	270	**270**	670
p3.3.h	300	**300**	16
p3.3.i	330	**330**	1311
p3.3.j	380	**380**	1770
p3.3.k	440	430	2075
p3.3.l	480	**480**	2570
p3.3.m	520	500	4647
p3.3.n	570	560	3846
p3.3.o	590	570	315
p3.3.p	640	**640**	4885
p3.3.q	680	640	4828
p3.3.r	710	**710**	4741
p3.3.s	720	710	5939
p3.3.t	760	720	70512
p3.4.a	20	**20**	< 1
p3.4.b	30	**30**	< 1
p3.4.c	90	**90**	< 1
p3.4.d	100	**100**	< 1
p3.4.e	140	**140**	1
p3.4.f	190	**190**	1
p3.4.g	220	**220**	6
p3.4.h	240	**240**	9
p3.4.i	270	**270**	89
p3.4.j	310	**310**	119
p3.4.k	350	**350**	147
p3.4.l	380	**380**	201
p3.4.m	390	**390**	274
p3.4.n	440	**440**	344
p3.4.o	500	480	372
p3.4.p	560	**560**	386
p3.4.q	560	**560**	419
p3.4.r	600	**600**	527
p3.4.s	670	**670**	607
p3.4.t	670	**670**	595
Average	414.67	410.67	5179.3

Table II displays the results obtained on the 33 instances of the second set (p2). Here, all the best known solutions in the literature are reached, and the average computation time is equal to 2.41 seconds. This indicates that these problems are

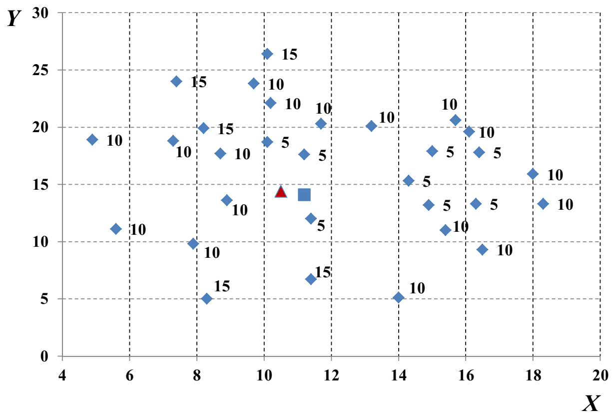

Fig. 3 Instance p.1.3.k where $n = 30$ (members), $m = 3$ and $T_{max} = 18.3$.

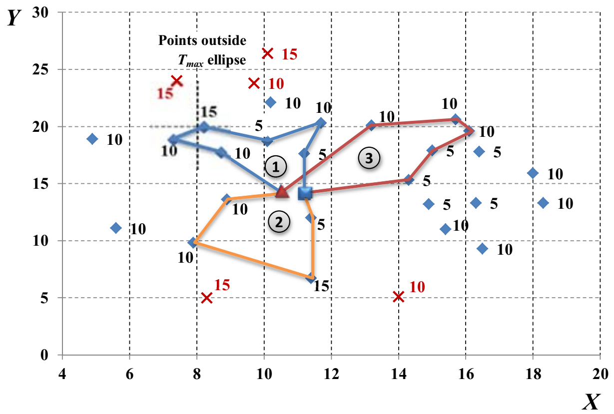

Fig. 4 Solution obtained by the proposed algorithm BSCBP on problem p1.3.k, total score = 135

easier to solve than those of set p1.

Finally, Table III contains the results obtained on the third set p3 that contains 60 instances. 47 best known solutions out of 60 are reached, which corresponds to 78%. The average computation time exceeds here 5000 seconds, this is because some instances instances are harder to solve. More precisely, the computation time exceeds 10000 seconds for instances p.3.2.i, p3.2.o, p.3.2.t, and attains 70512 seconds for p.3.3.t.

Fig. 3 shows an example of an instance where $|V| = 32$, (30 customers augmented with the start and end points), $m = 3$ members or vehicles, and $T_{max} = 18.3$. The score associated with each vertex is indicated. Points ▲ and ■ correspond to the start and end points respectively.

Fig. 4 indicates the solution obtained by the proposed algorithm BSCBP on instance p.1.3.k. The three obtained paths are:

- Path ① has a score $S(P_1) = 10+10+15+5+10+5 = 55$ and a total time $T(P_1) = 17.717$.
- Path ② : $S(P_2) = 5 + 15 + 10 + 10 = 40$, $T(P_2) = 17.975$.
- Path ③ : $S(P_3) = 10 + 10 + 10 + 5 + 5 = 40$, $T(P_3) = 17.803$.

So the total collected score is $S(\mathbb{P}) = 135$, this corresponds to the best known value in the literature for this instance.

Note also that there are five vertices that are outside the T_{max} ellipse. These vertices are then not considered when computing the solution.

V. Conclusion

In this work, a hybrid heuristic was presented in order to solve the team orienteering problem. The corresponding algorithm, denoted by BSCBP, is based on beam search that computes several paths in parallel in order to increase the probability to obtain "good ones" and a local optimization method that corresponds to the 3-opt strategy used to decrease the total time in the paths under creation. The results obtained on several set of problem instances show that the method is competitive since it reaches the best known results in the literature in 87% of cases.

References

[1] H. Bouly, D. C. Dang, and A. Moukrim, *A memetic algorithm for the team orienteering problem*, 4OR, vol. 8, 2010, pp. 49-70
[2] S. Boussier, D. Feillet, and M. Gendreau. *An exact algorithm for the team orienteering problem*, 4OR, vol. 5(3), 2007, pp. 211–230.
[3] S. E. Butt and D. Ryan, *An optimal solution procedure for the multiple tour maximum collection problem using column generation*, Computers & Operations Research, vol. 26(4), 1999, pp. 427–441.
[4] S. E. Butt and T. M. Cavalier, *A heuristic for the multiple tour maximum collection problem*, Computers & Operations Research, vol. 21(1), 1994, pp. 101–111.
[5] I. Chao, B. Golden, and E. Wasil, *The team orienteering problem*, European Journal of Operational Research, vol. 88(3), 1996, pp. 464–474.
[6] G. A. Croes, *A method for solving traveling salesman problems*, Operations Research, vol. 6, 1958, pp. 791–812.
[7] M. Desrochers, J. K. Lenstra, M. W. P. Savelsbergh, and F. Soumis, *Vehicle Routing with Time Windows: Optimization and Approximation*, Elsevier Science Publishers B.V. (North-Holland), 1988, pp. 65–84, in B.L. Golden and A.A. Assad (Editors), Vehicle Routing: Methods and Studies.
[8] Q. Hu and A. Lim, *An iterative three-component heuristic for the team orienteering problem with time windows*, European Journal of Operational Research, vol. 232, 2014, pp. 276–286.
[9] B. I. Kim, H. Li, and A. L. Johnson, *An augmented large neighborhood search method for solving the team orienteering problem*, Expert Systems with Applications, vol. 40(8), 2013, pp. 3065-3072.
[10] G. Laporte, *The Traveling Salesman Problem: An overview of exact and approximate algorithms*, European Journal of Operational Research, vol. 59, 1992, pp. 231–247.
[11] S. W. Lin, *Solving the team orienteering problem using effective multi-start simulated annealing*, Applied Soft Computing, vol. 13, 2013, pp. 1064–1073.
[12] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden, *The orienteering problem: A survey*, European Journal of Operational Research, vol. 209 (1), 2011, pp. 1-10.
[13] T. Vidal, T. G. Crainicc, M. Gendreau, and C. Prins, *Heuristics for multi-attribute vehicle routing problems: A survey and synthesis*, European Journal of Operational Research, vol. 231(1), 2013, pp. 1-21.