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 Travel 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 \( V = \{ v_0, \ldots, v_{n+1} \} \) 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 \( P = \{ P_1, \ldots, P_m \} \) 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).

\[
T(P_k) \leq T_{max}, \forall P_k \in \mathcal{P} \quad (1)
\]

The objective is then to maximize the sum of the collected scores in \( \mathcal{P} \), the set of all paths, each path \( P_k \in \mathcal{P} \) 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).

\[
\max S(\mathcal{P}) = \sum_{k=1}^{m} S(P_k) \quad (2)
\]

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 \(|\mathcal{P}| = m = 2\). 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 \( S(\mathcal{P}) = 15 + 20 = 35 \).
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:

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

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 $T_{\text{max}}$. Chao et al. [5] called this ellipse the “$T_{\text{max}}$ ellipse”. When the start point is the same than the end point, then we obtain a circle of diameter $T_{\text{max}}$.

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 $\omega$ nodes, where $\omega$ 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 $T_{\text{max}}$ ellipse. The output of BSCBP is the set of best paths found $P = \{ P_1, \ldots, P_m \}$ 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 $P_k \in P$ 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 \( T_{\text{max}} \) ellipse.

Output: The best paths found \( \mathbb{P} = \{P_1, \ldots, P_m\} \) that maximize the collected score.

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

The root node of the tree \( \eta_0 \) 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 \( \eta_0 \) and the current level \( \ell \) is set to 0 (line 6). The best node \( \eta^* \), 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 \( \ell \) of the tree, list \( B \) contains the nodes corresponding to the partial paths. Branching from node \( \eta_{t_j} \in \{P^+_j, P^-_j\} \in B \) 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 \( \sum |P^-_j| \) descendants. These offspring nodes are then stored in list \( B_{\text{off}} \) (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 \( B_{\text{off}} \) (line 14).

After that, if set \( P^- \) is empty (line 15) in a given node \( \eta_{t_j} \in B_{\text{off}} \), 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 \( \eta^* \) is updated if a greater score is obtained (lines 18) and the node is then removed from \( B_{\text{off}} \) (line 19).

The next step consists to filter list \( B_{\text{off}} \) in order to keep only the \( \omega \) best ones. This is done by sorting the nodes in \( B_{\text{off}} \) by using a given criterion \( \rho \), from the most important to the least important one, and then keeping the first (best) ones, the other nodes are removed from \( B_{\text{off}} \). 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 \( B_{\text{off}} \) are then assigned to \( B \) (line 23) and \( B_{\text{off}} \) 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 \( \eta^* \) (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 \( P_k \in \mathbb{P} \) for \( k = 1, \ldots, m \).

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 \( (v_i \rightarrow v_{i+1}), (v_j \rightarrow v_{j+1}) \), and \( (v_k \rightarrow v_{k+1}) \), and replace them by arcs \( (v_i \rightarrow v_j), (v_{i+1} \rightarrow v_k) \), and \( (v_{j+1} \rightarrow v_{k+1}) \) 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 \( (A \rightarrow B), (C \rightarrow D) \), and \( (E \rightarrow F) \) are replaced by arcs \( (A \rightarrow C), (B \rightarrow E) \), and
Algorithm 2: The 3-opt algorithm for decreasing the total time in a path.

**Input:** A path $P$ of total time $T(P)$

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

1. Improvement $\leftarrow$ true;
2. while (improvement = true) do
3. Improvement $\leftarrow$ false;
4. for each vertex $v_i \in P$ do
5. for each vertex $v_j \in P$ ($j \neq i - 1, j \neq i + 1$) do
6. for each vertex $v_k \in P$ ($k \neq j - 1, k \neq j + 1, k \neq i + 1, k \neq i - 1$) do
7. if $(t_{v_i,v_{i+1}} + t_{v_{i+1},v_j} + t_{v_k,v_{k+1}}) > (t_{v_i,v_j} + t_{v_{i+1},v_{j+1}})$ then
8. 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;
9. Improvement $\leftarrow$ true;
10. end if
11. end for
12. end for
13. end for
14. end while

(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 $T_{\text{max}}$ value.

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 $T_{\text{max}}$. For example for problem “p1.3.a”, there are 32 vertices including the start and end points, $m = 3$ and $T_{\text{max}} = 1.7$. And in problem “p1.3.k” we have exactly the same vertices (with the same scores) but the value of $T_{\text{max}}$ 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.
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 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 \(\omega\) (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 \(T_{\text{max}}\) 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 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 p1.3.k where \( n = 30 \) (members), \( m = 3 \) and \( T_{\text{max}} = 18.3 \).

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 10000 seconds, which is competitive since it reaches the best known results in the literature in 87% of cases.

REFERENCES


