The Set Orienteering Problem

Claudia Archetti[a,*], Francesco Carrabs[b], Raffaele Cerulli[b]

[a] *Department of Economics and Management, University of Brescia, Brescia, Italy*
[b] *Department of Mathematics, University of Salerno, Fisciano, Italy*

Abstract

In this paper we study the Set Orienteering Problem which is a generalization of the Orienteering Problem where customers are grouped in clusters and a profit is associated with each cluster. The profit of a cluster is collected only if at least one customer from the cluster is visited. A single vehicle is available to collect the profit and the objective is to find the vehicle route that maximizes the profit collected and such that the route duration does not exceed a given threshold. We propose a mathematical formulation of the problem and a matheuristic algorithm. Computational tests are made on instances derived from benchmark instances for the Generalized Traveling Salesman Problem with up 1084 vertices. Results show that the matheuristic produces robust and high-quality solutions in a short computing time.

Keywords: Routing, orienteering problem, matheuristic.

Introduction

Routing problems with profits received significant attention in recent years, as witnessed by the large literature surveyed in three recent papers, i.e., [3, 7] for routing problems with profits where the profit is associated with nodes of a graph, and [2] for problems where the profit is associated with arcs or edges of a graph. As witnessed by these three surveys, the literature on 'node' routing

*Corresponding author
Email addresses: `claudia.archetti@unibs.it` (Claudia Archetti), `fcarrabs@unisa.it` (Francesco Carrabs), `raffaele@unisa.it` (Raffaele Cerulli)

problems, i.e., the first of the two classes mentioned above, is much wider than the one on arc routing problems. In particular, among all node routing problems with profits studied in the literature, the most widely known is undoubtedly the Orienteering Problem (OP) where a profit is associated with each customer and the objective is to find a single vehicle tour maximizing the profit collected from visited customers and such that the duration of the tour does not exceed a maximum time limit. The profit of each customer can be collected at most once. The problem was first introduced in [13] and many variants of the problem have been studied, as described in [3, 7]. One of the most recent variants is the one presented in [1] where customers are grouped in clusters, a profit is associated with each cluster and it is collected only if all customers belonging to the cluster are visited. The authors called this problem the Clustered Orienteering Problem (COP). They present a mathematical formulation and two solution algorithms. They also describe practical applications related to the distribution of mass products where customers are retailers belonging to different supply chains.

In this paper we are interested in a variant of the OP which shows some analogies with the one studied in [1]. In particular, we study the problem where customers are grouped in clusters. A profit is associated with each cluster and is collected only if at least one customer from the cluster is visited. The objective is to find the vehicle route that maximizes the collected profit and such that the corresponding duration does not exceed a given threshold. We call this problem the Set Orienteering Problem (SOP). The SOP finds application in mass distribution products, as for the COP, where a different distribution plan is sought. In particular, consider the case where customers belong to different supply chains and the carrier stipulates contracts with chains. Then, instead of having to serve all retailers belonging to the chain with which the contract has been stipulated, as happens in the COP, in the SOP the carrier may choose to serve only one customer from the chain (and, implicitly, serving the entire quantity demanded by the chain). This way, the carrier may be able to offer a better price for the service. The inner distribution among all retailers in the chain will be then organized internally. Thus, the SOP presents an alternative to the distribution strategy applied in the COP which may be advantageous both for the carrier and for the chains. Another application arises when customers are clustered in areas and the service to each area is made by delivering the entire quantity required by all customers in the area to a single customer, the one that is visited. This happens also when private customers group together to reach large quantity orders, and thus hopefully a lower price. Typically, in this case, the delivery is made to a single location.

The contribution of this paper can be summarized as follows. We introduce the SOP, present a formal description and a mathematical formulation. We then propose a matheuristic algorithm for its solution which is tested on instances derived from benchmark instances for the Generalized Traveling Salesman Problem (GTSP) with up to 1084 vertices. In particular, we first show the performance of the algorithm on small instances by comparing the results obtained from the matheuristic with optimal solutions. In addition, we test the performance of the matheuristic on large instances for which the optimal solution is known. We then present an exhaustive study of the contribution of the MILP embedded in the matheuristic. The results show that the contribution of the MILP is more evident on large instances. Moreover, even if the MILP makes the overall algorithm slower, computing times remain reasonable even on the largest instances.

The paper is organized as follows. A formal description of the problem together with a mathematical formulation are presented in Section 1. The matheuristic algorithm is described in Section 2 while computational results are presented in Section 3. Finally, conclusions are drawn in Section 4.

1. Problem description and formulation

As the SOP is a generalization of the OP, we first provide a formal description of the OP.

The OP is defined on a complete directed graph $G = (V, A)$ where $V = \{0\} \cup C$. Vertex 0 represents the depot from which the vehicle starts and ends its tour. C is the set of customers. A profit p_i is associated with each customer $i \in C$ and is collected if and only if customer i is visited by the vehicle. Moreover, a cost c_{ij} is associated with each arc $(i, j) \in A$. The objective is to find the tour that maximizes the collected profit and such that the associated cost (or duration) does not exceed a maximum value T_{max}.

In this paper we study a variant of the OP which we call the *Set Orienteering Problem* (SOP). In the SOP, customers in C are grouped in clusters C_g with $g = 1, ..., l$ such that $\bigcup_{g=1}^{l} C_g = C$ and $C_g \cap C_h = \emptyset$, $\forall C_g, C_h \in \mathcal{P}$ where $\mathcal{P} = \{C_1, ..., C_l\}$ is the set of clusters. A profit p_g is associated with each cluster and is collected if and only if at least a customer $i \in C_g$ is visited in the tour. The profit of each cluster can be collected at most once. As in the OP, the objective is to find the tour that maximizes the collected profit and such that the associated cost does not exceed T_{max}. In the following we assume that costs c_{ij} satisfy the triangle inequality. In this case, as shown in [9], an optimal solution always exists where one vertex per cluster at most is visited. This property is used in the solution method presented in Section 2.

In order to present a mathematical formulation for the SOP, we need the following notation. For any subset of vertices $S \subset V$, we define $\delta^+(S) = \{(i, j) \in A : i \in S, j \notin S\}$ and $\delta^-(S) = \{(i, j) \in A : i \notin S, j \in S\}$. For the ease of presentation, in the following we will use the notation $\delta^+(i)$ and $\delta^-(i)$ when $S = \{i\}$. The decision variables are the following:

- y_i = binary variable equal to 1 if vertex $i \in V$ is visited by the vehicle, and 0 otherwise,
- x_{ij} = binary variable equal to 1 if arc $(i, j) \in A$ is traversed by the vehicle, and 0 otherwise,
- z_g = binary variable equal to 1 if the profit of cluster C_g is collected and 0 otherwise.

The mathematical programming formulation of the SOP is the following:

$$
\begin{aligned}
\max \quad & \sum_{j \in \mathcal{P}} p_g z_g && && (1) \\
\text{s.t.} \quad & \sum_{(i,j) \in \delta^+(i)} x_{ij} = y_i && i \in V, && (2) \\
& \sum_{(j,i) \in \delta^-(i)} x_{ji} = y_i && i \in V, && (3) \\
& \sum_{(i,j) \in \delta^+(S)} x_{ij} \geq y_h && S \subseteq V \setminus \{0\},\ h \in S, && (4) \\
& \sum_{(i,j) \in A} c_{ij} x_{ij} \leq T_{\max}, && && (5) \\
& z_g \leq \sum_{i \in C_g} y_i && C_g \in \mathcal{P}, && (6) \\
& y_i \in \{0,1\} && i \in V, && (7) \\
& z_g \in \{0,1\} && C_g \in \mathcal{P}, && (8) \\
& x_{ij} \in \{0,1\} && (i,j) \in A. && (9)
\end{aligned}
$$

The objective function (1) maximizes the collected profit. Constraints (2) and (3) ensure that one arc enters and one arc leaves each visited vertex. Subtours are eliminated through (4). Constraint (5) is the maximum duration constraint on the route while (6) imposes that the profit of cluster C_g is collected only if at least one customer $i \in C_g$ is visited in the tour. Finally, (7)-(9) are variable definitions.

Note that formulation (1)-(9) has an exponential number of subtour elimination constraints (4). A formulation with a polynomial number of constraints is obtained by introducing arc flow variables u_{ij}, representing the amount of flow crossing the edge (i, j), and substituting (4) with the following constraints:

$$
\begin{aligned}
\sum_{j \in V} u_{ji} - \sum_{j \in V} u_{ij} &= y_i && i \in V \setminus \{0\}, && (10) \\
u_{ij} &\leq (n-1) x_{ij} && (i,j) \in A, && (11) \\
y_0 &= 1 && && (12) \\
u_{ij} &\geq 0 && (i,j) \in A, && (13)
\end{aligned}
$$

where $n = |V|$. We note that this formulation of subtour elimination constraints has been proposed in [6] for the Traveling Salesman Problem (TSP) and its performance has been recently assessed in [11].

2. A matheuristic for the SOP

In this section we describe the heuristic algorithm we have designed for the solution of the SOP. It is a matheuristic algorithm which is composed by the following two phases:

1. *Phase 1: Construction of an initial solution.*
2. *Phase 2: Tabu search.*

It is a matheuristic algorithm as the tabu search makes use of a MILP formulation when it struggles in finding a new non-tabu feasible solution. In the following we refer to the algorithm as MASOP - a MAtheuristic for the SOP.

We now describe in details the two phases. We define the following notation. Given a tour T, we denote as $C(T) \subseteq \mathcal{P}$ the set of clusters visited in T, i.e., the set of clusters for which at least one vertex is visited in T, while $p(T)$ and $c(T)$ are the profit and the cost associated with T, respectively. T is a feasible tour if and only if $c(T) \leq T_{max}$. Moreover, given a vertex $i \in V \backslash \{0\}$, let $C(i)$ be the cluster to which i belongs. On the other hand, given a cluster C_g, let $V(C_g)$ be the set of all vertices belonging to C_g.

2.1. Phase 1: Construction of an initial solution

MASOP constructs an initial feasible solution through a simple greedy algorithm which inserts clusters in the tour as long as this is feasible. In particular, it starts with a path T visiting vertex 0 only. Let u be the last vertex visited in T and let C_f be the cluster in $\mathcal{P} \backslash C(T)$ with the highest profit, i.e, $C_f = argmax_{C_g \in \mathcal{P} \backslash C(T)} \{p_g\}$. Then, the procedure attaches at the end of T the vertex $v \in C_f$ such that $c_{uv} + c_{v0}$ is minimum, if $c(T) + c_{uv} + c_{v0} \leq T_{max}$. If no such vertex exists, the procedure considers the cluster with the second highest profit and iterates. If no vertex can be feasibly inserted in T, the procedure stops and u is linked with 0. This way, T becomes a tour.

Once a feasible solution is constructed as described above, before starting Phase 2, MASOP applies the following procedure which is aimed at improving the cost associated with tour T. The idea is to maintain the same set $C(T)$ of clusters visited, and thus the same associated profit $p(T)$, while reducing cost $c(T)$. The procedure consists of two steps.

In the first step, the procedure determines, for each cluster of $C(T)$, which vertex has to be visited in order to obtain the minimum cost tour provided that the clusters in $C(T)$ are visited according to the sequence defined in T. This problem can be solved in polynomial time by solving a shortest path problem as shown in [8, 12, 4]. Note that the problem can be solved as a shortest path thanks to the fact that the triangle inequality holds, as shown in [8]. Once the first step is terminated, we apply the second step which aims at optimizing the sequence of visiting the vertices included in T. In particular, let $V(T)$ be the set of vertices visited in T. The second step consists in applying the Lin-Kernighan algorithm [10] for the solution of the Traveling Saleman Problem to the set $V(T)$. The procedure iterates between the first and the second step as long as an improvement is found. The sketch of the procedure is shown in Algorithm 1. We call the procedure *Tour Improvement*.

Algorithm 1 Tour Improvement

```
1: Input: Tour T.
2: Output: Tour T_best.
3: T_best ← T.
4: T ← Solve Shortest Path(T).
5: T ← Solve Lin-Kernighan (V(T)).
6: if (c(T_best) > c(T))
7:     T_best ← T. Return to 4.
8: else
9:     STOP.
10: Return T_best.
```

2.2. Phase 2: Tabu search

Phase 2 aims at improving the initial solution constructed in Phase 1 through a tabu search which is composed by the following procedures:

- $ExploreNeighborhood(T, TL)$: this procedure explores the neighborhood of the solution represented by T on the basis of the tabu list TL and is explained in Section 2.2.1.

- $MIPMove(T)$: this function explores a different neighborhood which is based on the solution of a MILP. No tabu list is considered. It is called when function $ExploreNeighborhood(T, TL)$ does not find any feasible non-tabu solution. $MIPMove(T)$ is described in Section 2.2.2.

- $Shake(T)$: this is a diversification procedure which destroys the current solution T by removing a number of clusters visited in T at random. It is described in Section 2.2.3. It uses two operators: $SoftShake(T)$ and $HardShake(T)$.

A sketch of the tabu search algorithm is provided in Algorithm 2.

The tabu search is started with the initial feasible solution provided by Phase 1 (line 1). The main loop of the algorithm is in lines 5–44. The algorithm continues until a maximum number of iterations without improvement is reached (line 5). The counter of iterations without improvement is set to 0 any time a new best solution is found (line 21 and 31) and is decreased by 1 any time the new solution improves the current one (line 11).

The main loop works as follows. First, the $ExploreNeighborhood(T, TL)$ procedure is called (line 8). If it succeeds in finding a non-tabu feasible solution, then the move is implemented (line 13) and the tabu list is updated (line 14). The neighborhood is entirely explored and the best move is implemented. In addition, every α iterations the Tour Improvement procedure (Algorithm 1) is applied to the current tour T (line 17). If $ExploreNeighborhood(T, TL)$ fails in finding a new non-tabu feasible solution, then $MIPMove(T)$ is invoked (line 25) only if $MIPInvocable = true$. $MIPInvocable$ is a binary

Algorithm 2 Tabu Search

```
Input: Tour T. \\ Provided by Phase 1
Output: Tour T_best.
T_best ← T.
numIterations ← 0; MIPInvocable ← true.
while numIterations ≤ maxIterations do
    numIterations++.
    ShakeInvocable ← true.
    T' ← ExploreNeighborhood(T, TL).
    if T' ≠ null then
        if p(T') > p(T) then
            numIterations−−.
        end if
        T ← T'.
        Update TL.
        MIPInvocable ← true.
        if numIterations%α = 0 then
            T ← Tour Improvement (T).
        end if
        if p(T) > p(T_best) then
            T_best ← Tour Improvement (T).
            numIterations ← 0.
        end if
    else
        if (MIPInvocable = true) then
            T' ← MIPMove(T).
            if p(T') > p(T) then
                T ← T'.
                ShakeInvocable ← false; MIPInvocable ← false.
                if p(T') > p(T_best) then
                    T_best ← Tour Improvement (T').
                    numIterations ← 0.
                end if
            end if
        end if
        if (ShakeInvocable = true) then
            MIPInvocable ← true.
            if p(T) ≥ p(T_best) − β ∗ p(T_best) then
                T ← SoftShake(T).
            else
                T ← HardShake(T).
            end if
        end if
    end if
end while
Return T_best.
```

variable that checks whether the $MIPMove(T)$ procedure has already been applied to the current tour T (in this case $MIPInvocable = false$) or not ($MIPInvocable = true$). A similar role has variable $ShakeInvocable$ for the $Shake(T)$ function. If $MIPMove(T)$ improves the current solution, then both $MIPInvocable$ and $ShakeInvocable$ are set to false as the idea is to first direct the search to the improvement of the new solution just found (line 28). Finally, there is the shaking phase which is invoked only if $ShakeInvocable = true$, i.e., when $ExploreNeighborhood(T, TL)$ fails in finding a new non-tabu solution and $MIPMove(T)$ does not improve the current solution. Then, if the profit of the current tour T is not lower than β times the profit of the best tour, $SoftShake(T)$ is applied. Otherwise, the algorithm applies $HardShake(T)$ (line 37).

We now explain the three main procedures $ExploreNeighborhood(T, TL)$, $MIPMove(T)$ and $Shake(T)$ in detail.

2.2.1. Procedure $ExploreNeighborhood(T, TL)$

Given the current solution represented by tour T, the neighborhood of T, which we call $N(T)$, is defined by the following moves:

- **Insert.** For all clusters $C_g \in \mathcal{P} \backslash C(T)$, the operator evaluates the insertion of C_g in T. The insertion works as follows. Let $< v_1, \ldots, v_k >$ be the sequence of vertices visited in T, where $v_1 = v_k = 0$. Starting from v_1 the operator determines the vertex $i \in C_g$ for which $c_{v_1 i} + c_{i v_2}$ is minimum. The insertion is evaluated for all insertion points, i.e., for all v_j with $j = 1, ...k - 1$ and the best one is retained only if it leads to a new cost not greater than T_{max}.

- **Swap.** For each pair of clusters $C_g \in \mathcal{P} \setminus C(T)$ and $C_h \in C(T)$, the operator evaluates the exchange of them, i.e., inserting C_g in T and removing C_h. The procedure is done by first evaluating the removal of C_h and then evaluating the insertion of C_g as done by the operator **Insert**. The removal of C_h is done by simply joining the predecessor of the vertex in C_h visited in T with its successor.

An iteration of the tabu search consists in evaluating all the neighbor solutions in $N(T)$. Infeasible solutions are discarded and the best non-tabu solution is chosen as the new solution, thus becoming the current solution at the next iteration. Solutions are evaluated through a hierarchical objective function where the first objective consists of the profit associated with the solution and the second is the associated cost. Thus, the best solution in terms of total profit is chosen. In case of ties, the solution with the lowest cost is chosen.

The tabu list is defined as follows. When a cluster is removed (inserted) from the current solution T, then it is tabu to reinsert (remove) it for a number of iterations equal to

$$rand(\lceil l\lambda \rceil) \tag{14}$$

where $rand(\lceil l\lambda \rceil)$ is a function that returns a random integer number in $(0, \lceil l\lambda \rceil]$, $l = |\mathcal{P}|$ and λ is a positive parameter. Moreover, each time a new best solution is found, procedure *Tour Improvement* is called to reduce the cost $c(T)$. *Tour Improvement* is also called every α iterations of the tabu search and applied on the current solution, independently of the fact that this is a new best solution or not.

2.2.2. Procedure $MIPMove(T)$

$MIPMove(T)$ is called if $ExploreNeighborhood(T, TL)$ fails in finding a new feasible non-tabu solution. The idea is to apply a move with a broader range of possible changes to the current solution T and, thus, a higher probability of finding a new feasible solution. No tabu list is considered. It works as follows.

First, a set of clusters visited in T is removed from T. The number δ of removed clusters is chosen randomly in $[5\%|T|; 15\%|T|]$. This interval gives the possibility of exploring a larger neighborhood than $ExploreNeighborhood(T, TL)$ without destroying the current solution T. In order to choose the clusters to be removed, clusters in T are ordered on the basis of a non-decreasing value of the ratio

$$\frac{p_g}{rs_g}$$

where rs_g is the removal saving obtained by removing the vertex $v \in C_g \cap T$ from T (i.e., joining its predecessor with its successor). If $rs_g = 0$ then we fix it to 1. This last situation occurs when $v \in C_g$ is on the segment connecting its predecessor and its successor in T. The subset composed by the first $\delta * 1.5$ clusters of the ordered list is considered and then δ clusters are chosen randomly from this subset. This way, a tour T' on the remaining clusters is found. By multiplying the parameter δ by 1.5, we avoid to always select the first δ clusters from the sorted list and we introduce randomness in this selection.

Then, a MILP is solved to insert non-visited clusters in T'. Let us define $D = \mathcal{P} \backslash C(T')$ as the set of clusters not visited in T' and Γ be the set of all subsets of D composed by one or two clusters. Moreover, let $< v_1, \dots, v_k >$ be the sequence of vertices visited in T', with $v_1 = v_k = 0$. For each set $\gamma \in \Gamma$, we compute the cost of inserting γ between v_i and v_{i+1}, $i = 1, ..., k-1$, which we denote $\Delta_{\gamma i}$. Cost $\Delta_{\gamma i}$ is computed as follows. In case γ is formed by a single cluster C_g, then $\Delta_{\gamma i}$ corresponds to the length of the shortest path visiting, in sequence, v_i - a vertex in C_g - v_{i+1}. In case γ is formed by two clusters C_g and C_h, then $\Delta_{\gamma i}$ corresponds to the minimum between the length of the shortest paths v_i - a vertex in C_g - a vertex in C_h - v_{i+1} and v_i - a vertex in C_h - a vertex in C_g - v_{i+1}. Note that we do not consider subsets of clusters of cardinality greater than 2 as this would be computationally too expensive. Finally, let $a_{\gamma g} = 1$ if set $\gamma \in \Gamma$ contains cluster $C_g \in D$, 0 otherwise. The following MILP is then solved:

$$
\begin{aligned}
\max \quad & \sum_{\gamma \in \Gamma} \sum_{i=1}^{k-1} p(\gamma) w_{\gamma i} && (15)\\
\text{s.t.} \quad & \sum_{\gamma \in \Gamma} \sum_{i=1}^{k-1} a_{\gamma g} w_{\gamma i} \leq 1 \qquad C_g \in D, && (16)\\
& \sum_{\gamma \in \Gamma} w_{\gamma i} \leq 1 \qquad i = 1, ..., k-1, && (17)\\
& c(T') + \sum_{\gamma \in \Gamma} \sum_{i=1}^{k-1} \Delta_{\gamma i} w_{\gamma i} \leq T_{max}, && (18)\\
& w_{\gamma i} \in \{0, 1\} \qquad \gamma \in \Gamma, \quad k = 1, ..., k-1 && (19)
\end{aligned}
$$

where $w_{\gamma i}$ is a binary variable which takes value 1 if set γ is inserted after vertex v_i, 0 otherwise. The objective function (15) aims at maximizing the profit of the clusters inserted. Constraints (16) state that each cluster can be inserted at most once while (17) establish that at most one element from Γ can be inserted between two consecutive vertices v_i and v_{i+1}. Finally, (18) is the maximum duration constraint.

The solution of (15)–(19) provides a new tour T''. If $p(T'') > p(T)$, then the procedure is iterated. A sketch of $MIPMove(T)$ is provided in Algorithm 3.

Algorithm 3 $MIPMove(T)$

1: Input: Tour T.
2: Output: Tour T''.
3: $T' \leftarrow$ remove δ clusters from T.
4: $T'' \leftarrow$ solve (15)–(19) on T'.
5: **if** $p(T'') > p(T)$ **then**
6:     $T \leftarrow$ Tour Improvement (T'').
7:     Return to line 3.
8: **end if**
9: Return T.

2.2.3. *Procedure Shake(T)*

$Shake(T)$ aims at destroying the current solution and directing the search to a different part of the solution space. It removes at random a certain number of clusters from T. The only difference between $SoftShake(T)$ and $HardShake(T)$ is the number of clusters removed. In particular, in $SoftShake(T)$ this number is chosen randomly in $[5\%|T|; 15\%|T|]$ while in $HardShake(T)$ it is in $[30\%|T|; 40\%|T|]$.

3. Computational tests

In this section we present the computational results of the tests we made in order to evaluate the performance of MASOP. The algorithm was coded in C++ on an OSX platform (Imac late 2012), running on an Intel Core i7 2.8 GHz processor with 16 GB of RAM. The mathematical formulations were solved using the ILOG Concert Technology library and CPLEX 12.6.

In the following section we describe how we generated the instances for the SOP while computational results are illustrated in Section 3.2.

3.1. *Test instances*

As the SOP has not been studied previously in the literature, no benchmark instances exist. Thus, we generated them by adapting instances for the Generalized Traveling Salesman Problem (GTSP) proposed in [5]. The GTSP is the problem where customers are divided in cluster and the objective is to find the shortest cycle visiting at least one customer per cycle. In particular, among all instances proposed in [5], we select the instances for which the distance is defined as the Euclidean distance between customers' coordinates. These instances have a number of vertices ranging from 52 to 1084 and a number of clusters equal to $\sim 20\%$ of the number of vertices. They are 51 in total. Concerning the number of customers, we classify the instances in two groups: *small* instances with up to 198 customers and *large* instances with at least 200 customers. We adapted these instances to the SOP as follows. We maintain the data related to customers locations. As no depot is defined in the GTSP, we defined the depot as follows. We took the first node in the node list and set it to be the depot. Thus, we removed it from the cluster it was inserted into in the GTSP instance and we inserted it in cluster 0 which contains the depot only. Then, we needed to generate the profits p_g for $C_g \in \mathcal{P}$ and T_{max}. We generated them as follows.

- p_g: we used two rules. The first rule sets the profit of each cluster C_g equal to $|C_g|$. The second rule sets the profit of a node j equal to $1 + (7141j+73) mod(100)$ in order to obtain pseudo-random profits. The profit of a cluster in then obtained by summing up the profit of all the nodes belonging to it. The same rules are used in [1] for the clustered OP. In the following we call g_1 the first rule and g_2 the second rule.

- T_{max}: we set it to $\omega GTSP^*$ where $GTSP^*$ is the cost of the best known solution value of the GTSP (taken from [5]) and ω has been set to 0.4, 0.6, 0.8.

Thus, in total we had 306 instances which form the **Set 1** instances. In addition, we generated a second set of instances, which we called **Set 2**, starting from the same instances and following the same procedure described above. However, we changed the clusters defined in the GTSP instances as follows. We maintain the same number of clusters defined in the GTSP instances and assigned the customers at random to clusters. The value of $GTSP^*$ is not available for these new instances so we kept the one used for instances of Set 1.

3.2. Computational results

We performed preliminary tests, by considering all the instances of Set 1, in order to fine-tune the parameters used in MASOP. The resulting values are the following:

- $MaxIterations = 500$: it corresponds to the maximum number of iterations without improvement. It determines the stopping condition of the tabu search.

- $\lambda = 0.1$: it determines the tabu length.
- $\alpha = 5$: it is related to the frequency of applying the *Tour Improvement* procedure.
- $\beta = 8\%$: it establishes whether $SoftShake(T)$ or $HardShake(T)$ is applied.

We now present the results of our tests. In the following, all computing times are expressed in seconds. We made two kind of experiments. The first one is aimed at verifying the efficacy of MASOP by comparing the value of the solutions it provides with the optimal solutions obtained *i)* by solving the mathematical formulation for the SOP presented in Section 1 and *ii*) by considering the special case when $\omega = 1$ for which the value of the optimal solution corresponds to the sum of the profit of all clusters. Regarding the mathematical formulation, we used inequalities (10)–(13) to model subtour elimination constraints. The formulation was solved through CPLEX 12.6. The maximum CPU time was set to 9 hours. Only small size instances were tested as no optimal solutions is obtained for larger ones.

Results are shown in Tables 1 and 2. In Table 1, the first block of rows is related to instances of Set 1 while the second block is related to instances of Set 2. The table is organized as follows. The first four columns report data on the instances: the name of the instance, the total number of vertices (n), the value of ω and the kind of profit. The following two columns refer to the computation of the optimal solution: value of the optimal solution and the computational time taken by CPLEX to compute it. Finally, the last three columns are related to MASOP solutions: solution value, computational time and percentage gap with respect to the optimal solution. From Table 1 it is possible to notice that MASOP always finds the optimal solution of Set 1 instances while it fails to find the optimal solution on three instances of Set 2 with a maximum gap of 8.11%. The computational time spent by MASOP to solve these instances is always lower than 15 seconds.

					MIP		MASOP		
	Instance	**n**	ω	p_g	**Opt**	**Time**	**Sol**	**Time**	**Gap**
Set 1	11berlin52	52	0.4	g_1	37	47.07	37	1.75	0.00%
	11berlin52	52	0.4	g_2	1829	65.96	1829	1.70	0.00%
	11berlin52	52	0.6	g_1	43	777.88	43	2.40	0.00%
	11berlin52	52	0.6	g_2	2190	1532.91	2190	2.64	0.00%
	11berlin52	52	0.8	g_1	47	2648.04	47	7.17	0.00%
	11berlin52	52	0.8	g_2	2384	3833.50	2384	6.61	0.00%
	11eil51	51	0.4	g_1	24	39.72	24	1.85	0.00%
	11eil51	51	0.4	g_2	1279	40.13	1279	1.97	0.00%
	11eil51	51	0.6	g_1	39	34.64	39	5.13	0.00%
	11eil51	51	0.6	g_2	1911	204.65	1911	4.74	0.00%
	11eil51	51	0.8	g_1	43	1586.67	43	2.30	0.00%
	11eil51	51	0.8	g_2	2114	1520.67	2114	1.93	0.00%
	14st70	70	0.4	g_1	33	9666.29	33	4.43	0.00%
	14st70	70	0.4	g_2	1672	4396.77	1672	4.35	0.00%
	14st70	70	0.8	g_1	65	18227.23	65	8.80	0.00%
	14st70	70	0.8	g_2	3355	30851.18	3355	7.89	0.00%
	16eil76	76	0.4	g_1	40	4987.09	40	3.88	0.00%
	16eil76	76	0.4	g_2	2223	4939.08	2223	4.73	0.00%
	16eil76	76	0.6	g_1	59	29565.85	59	2.40	0.00%
	16eil76	76	0.6	g_2	3119	21127.41	3119	6.28	0.00%
Set 2	11berlin52_RND	52	0.4	g_1	50	43.20	50	6.33	0.00%
	11berlin52_RND	52	0.4	g_2	2584	60.84	2584	6.11	0.00%
	11berlin52_RND	52	0.6	g_1	51	1.52	51	6.73	0.00%
	11berlin52_RND	52	0.6	g_2	2608	1.14	2608	6.63	0.00%
	11eil51_RND	51	0.4	g_1	37	9.12	**34**	0.82	8.11%
	11eil51_RND	51	0.4	g_2	1929	7.44	1929	1.30	0.00%
	11eil51_RND	51	0.6	g_1	50	6.28	50	5.27	0.00%
	11eil51_RND	51	0.6	g_2	2575	54.73	2575	5.25	0.00%
	14st70_RND	70	0.4	g_1	56	5316.83	56	5.60	0.00%
	16eil76_RND	76	0.4	g_1	51	2498.79	51	4.00	0.00%
	16eil76_RND	76	0.6	g_1	74	993.13	**69**	0.86	6.76%
	16eil76_RND	76	0.6	g_2	3728	1737.65	**3713**	5.76	0.40%
	16pr76_RND	76	0.4	g_1	70	6536.81	70	3.09	0.00%
	16pr76_RND	76	0.4	g_2	3550	5806.05	3550	7.39	0.00%
	16pr76_RND	76	0.6	g_1	74	1370.58	74	12.57	0.00%
	20kroE100_RND	100	0.4	g_2	4614	7865.33	4614	8.83	0.00%
	20rat99_RND	99	0.4	g_1	73	3103.25	73	2.34	0.00%
	20rat99_RND	99	0.4	g_2	3624	4408.47	3624	2.15	0.00%
	21lin105_RND	105	0.6	g_1	104	2085.34	104	11.11	0.00%
	21lin105_RND	105	0.6	g_2	5228	2034.79	5228	11.85	0.00%
	22pr107_RND	107	0.6	g_1	106	1021.05	106	12.53	0.00%
	22pr107_RND	107	0.6	g_2	5363	2145.44	5363	11.89	0.00%

Table 1: Comparison with optimal solutions on small instances with $\omega < 1$.

To further investigate the effectiveness of MASOP on large instances too, we carried out computational tests on Set 1 instances by using $\omega = 1$, i.e. T_{max} is equal to the cost of the best known GTSP solution. This means that there is at least a tour that can visit all the clusters without violating the threshold T_{max} and then the optimal profit coincides with the sum of the profits of all the clusters. In other words, we apply MASOP to solve the GTSP. Obviously we do not expect to obtain results equal to the specific metaheuristics created “ad hoc” for this problem but the gap between the optimal solution values and the ones found by MASOP allow us to have an idea about the effectiveness of our algorithm even on large instances.

Results are shown in Table 2 where the first block of rows is related to the small instances while the second block is related to the large instances. The first two columns report the name of the instance and the total number of vertices. The next 8 columns are divided in two blocks, related to the profit rules g_1 and g_2, reporting: the optimal solution value (Opt), the MASOP solution value (Sol), MASOP computational time (Time) and percentage gap with respect to the optimal solution (Gap). On small instances with profit g_1, the percentage gap is lower than 3% on 19 out of 27 instances while the peak is equal to 5.1% and occurs on the instance 20rat99. Anyway, this is the only case for which the gap is higher than 5% while the average gap is equal to 2.15%. Surprisingly, the results are much better on the large instances. Indeed, the gap is always lower than 3% and the peak is equal to 2.23% (45tsp225). It is worth noting that, in these instances, the gap is lower than 1% on 17 out of 24 times and that the average gap is 0.81%. In our opinion, this situation occurs because on the large instances it is more difficult to find an optimal solution for the Generalized TSP and then the best known solution, provided in literature for these instances, is not the optimal one. For this reason, the value of T_{max} is less restrictive on the large instances with respect to the small instances. As a consequence, the operators of MASOP are more effective thanks to the less binding duration constraint imposed during the construction or the modification of the tour.
We observed a similar behaviour on the instances with profit type g_2. In partic-

			g_1				g_2			
				MASOP				MASOP		
	Instance	**n**	**Opt**	**Sol**	**Time**	**Gap**	**Opt**	**Sol**	**Time**	**Gap**
Small	11berlin52	52	51	50	8.54	1.96%	2608	2580	8.54	1.07%
	11eil51	51	50	48	7.03	4.00%	2575	2426	6.93	5.79%
	14st70	70	69	68	8.02	1.45%	3513	3488	8.25	0.71%
	16eil76	76	75	74	8.17	1.33%	3800	3780	8.23	0.53%
	16pr76	76	75	74	8.57	1.33%	3800	3765	10.58	0.92%
	20kroA100	100	99	96	10.53	3.03%	5008	4868	11.48	2.80%
	20kroB100	100	99	98	13.40	1.01%	5008	4916	10.73	1.84%
	20kroC100	100	99	97	10.80	2.02%	5008	4882	11.23	2.52%
	20kroD100	100	99	96	10.30	3.03%	5008	4838	8.86	3.39%
	20kroE100	100	99	96	9.14	3.03%	5008	4887	9.95	2.42%
	20rat99	99	98	93	7.57	5.10%	5007	4721	8.09	5.71%
	20rd100	100	99	97	10.38	2.02%	5008	4929	9.60	1.58%
	21eil101	101	100	97	8.81	3.00%	5050	4953	20.10	1.92%
	21lin105	105	104	102	8.05	1.92%	5228	5157	8.38	1.36%
	22pr107	107	106	101	8.62	4.72%	5363	5109	8.28	4.74%
	25pr124	124	123	121	11.30	1.63%	6232	6173	11.92	0.95%
	26bier127	127	126	125	15.97	0.79%	6333	6314	16.25	0.30%
	26ch130	130	129	127	10.21	1.55%	6503	6412	9.73	1.40%
	28pr136	136	135	134	10.24	0.74%	6850	6841	12.24	0.13%
	29pr144	144	143	141	17.35	1.40%	7242	7195	22.01	0.65%
	30ch150	150	149	144	9.59	3.36%	7533	7315	12.32	2.89%
	30kroA150	150	149	145	11.29	2.68%	7533	7361	13.76	2.28%
	30kroB150	150	149	148	15.25	0.67%	7533	7445	15.14	1.17%
	31pr152	152	151	147	18.22	2.65%	7658	7422	17.77	3.08%
	32u159	159	158	157	15.51	0.63%	8037	7991	22.87	0.57%
	39rat195	195	194	189	13.04	2.58%	9863	9558	11.02	3.09%
	40d198	198	197	196	36.76	0.51%	9997	9934	25.74	0.63%
AVG					**11.95**	**2.15%**			**12.59**	**2.02%**
Large	40kroa200	200	199	198	22.75	0.50%	10058	10010	24.54	0.48%
	40krob200	200	199	198	19.86	0.50%	10058	9990	28.58	0.68%
	45ts225	225	224	221	34.67	1.34%	11308	11187	26.18	1.07%
	45tsp225	225	224	219	16.70	2.23%	11308	11103	16.44	1.81%
	46pr226	226	225	224	26.94	0.44%	11375	11368	26.44	0.06%
	53gil262	262	261	258	27.20	1.15%	13193	13050	25.92	1.08%
	53pr264	264	263	262	34.02	0.38%	13302	13277	36.94	0.19%
	56a280	280	279	273	33.52	2.15%	14178	13971	37.03	1.46%
	60pr299	299	298	296	31.34	0.67%	15107	15005	36.75	0.68%
	64lin318	318	317	315	43.42	0.63%	16037	16013	77.26	0.15%
	80rd400	400	399	397	76.68	0.50%	20158	20055	48.06	0.51%
	84fl417	417	416	415	103.55	0.24%	21048	21030	114.65	0.09%
	88pr439	439	438	437	157.96	0.23%	22177	22110	132.76	0.30%
	89pcb442	442	441	440	129.54	0.23%	22323	22300	94.98	0.10%
	99d493	493	492	490	120.84	0.41%	24862	24827	153.07	0.14%
	115rat575	575	574	562	91.23	2.09%	29033	28497	65.91	1.85%
	115u574	574	573	571	204.52	0.35%	28957	28888	212.74	0.24%
	131p654	654	653	652	356.02	0.15%	32997	32991	360.06	0.02%
	132d657	657	656	649	126.14	1.07%	33188	32974	155.92	0.64%
	145u724	724	723	716	99.60	0.97%	36532	36288	116.69	0.67%
	157rat783	783	782	767	279.23	1.92%	39517	38953	145.54	1.43%
	201pr1002	1002	1001	994	304.91	0.70%	50583	50453	992.74	0.26%
	212u1060	1060	1059	1057	873.47	0.19%	53548	53450	798.51	0.18%
	217vm1084	1084	1083	1078	489.48	0.46%	54712	54642	655.52	0.13%
AVG					**154.32**	**0.81%**			**182.63**	**0.59%**

Table 2: Comparison with optimal solutions on small and large instances with $\omega = 1$.

ular, on the small instances, the percentage gap is lower than 3% on 21 out of 27 instances while the gap is higher than 5% only in two cases with a peak equal to 5.79% on instance 11eil51. Again, better results are obtained by MASOP on the large instances where the gap is always lower than 2% and in 18 out of 24 cases it is lower than 1%. The peak is equal to 1.85% and occurs on instance 115rat575 and the average gap is equal to 0.59%.

Regarding the computational time, MASOP solves the small instances in less than 37 seconds with an average time equal to around 12 seconds for both profit rules g_1 and g_2. On large instances, the computational time significantly increases with an average time equal to 154 seconds for g_1 and 182 seconds for g_2. Moreover, we observed a peak equal to 837 and 992 seconds for g_1 and g_2, respectively.

According to the results reported in Table 1 and Table 2, we can conclude that MASOP provides robust and reliable results related to high quality solutions. Finally, we did not carry out the computational tests with $\omega = 1$ for the Set 2 because MASOP found solutions where all the clusters are visited in 100 out of 102 instances with $\omega = 0.8$.

Next we move to larger instances, with $\omega < 1$, for which no optimal solution is available. The aim of this second set of experiments is to show the effectiveness of the different procedures embedded in MASOP. In particular, we aim at showing the effectiveness of the tabu search in improving the initial solution and, more specifically, the effectiveness of Procedure $MIPMove(T)$. In order to do that, we run two versions of MASOP: the complete version as described in Section 2 and a version where Procedure $MIPMove(T)$ is removed. We call this second version HSOP for *Heuristic for the SOP*. In fact, this algorithm is no more a matheuristic algorithm and is a standard heuristic.

Results are shown in Table 3. The table is divided in two blocks of rows related to instances of Set 1 and 2. We report average results on the instances clustered according to their number of customers, the value of ω and the rule used to generate profits. We report average results over instances belonging to the same cluster. The first three columns of the table report: the class of

	Size	p_g	ω	T_S	T_B	T_B *vs* T_S	HSOP #best	HSOP profit	HSOP Gap%	MASOP #best	MASOP profit	MASOP Gap%
Set 1	**Small**	g_1	0.4	35.78	53.33	32.91%	27/27	53.33	0.00%	27/27	53.33	0.00%
			0.6	45.21	80.70	43.98%	25/27	80.62	0.10%	27/27	80.70	0.00%
			0.8	53.96	99.55	45.80%	26/27	99.55	0.00%	26/27	99.51	0.04%
		avg				*40.89%*	*26/27*		*0.03%*	*26.6/27*		*0.01%*
		g_2	0.4	2011.14	2681.66	25.00%	27/27	2681.66	0.00%	26/27	2681.22	0.02%
			0.6	2395.74	4059.70	40.99%	27/27	4059.70	0.00%	27/27	4059.70	0.00%
			0.8	2911.96	5037.37	42.19%	26/27	5037.37	0.00%	26/27	5034.03	0.07%
		avg				*36.06%*	*26.6/27*		*0.00%*	*26.3/27*		*0.03%*
	avg					*38.48%*	*26.3/27*		*0.02%*	*26.5/27*		*0.02%*
	Large	g_1	0.4	81.58	245.83	66.81%	19/24	245.83	0.00%	21/24	244.66	0.48%
			0.6	111.33	361.62	69.21%	13/24	361.33	0.08%	20/24	361.62	0.00%
			0.8	126.54	447.66	71.73%	16/24	446.25	0.31%	19/24	447.66	0.00%
		avg				*69.25%*	*16/24*		*0.13%*	*20/24*		*0.16%*
		g_2	0.4	4046.12	12482.95	67.59%	15/24	12469.00	0.11%	21/24	12482.95	0.00%
			0.6	5365.70	18433.33	70.89%	12/24	18394.29	0.21%	20/24	18433.33	0.00%
			0.8	6744.45	22797.41	70.42%	14/24	22797.41	0.00%	17/24	22768.25	0.13%
		avg				*69.63%*	*13.6/24*		*0.11%*	*19.3/24*		*0.04%*
	avg					*69.44%*	*14.8/24*		*0.12%*	*19.6/24*		*0.10%*
avg						*53.96%*	*20.5/24*		*0.07%*	*23/24*		*0.06%*
Set 2	**Small**	g_1	0.4	56.35	95.59	41.05%	26/27	95.59	0.00%	25/27	95.59	0.00%
			0.6	69.46	112.48	38.25%	24/27	112.48	0.00%	25/27	112.18	0.27%
			0.8	80.64	115.70	30.30%	27/27	115.70	0.00%	27/27	115.70	0.00%
		avg				*36.53%*	*25.6/27*		*0.00%*	*25.6/27*		*0.09%*
		g_2	0.4	2923.74	4825.74	39.41%	23/27	4811.18	0.30%	26/27	4825.74	0.00%
			0.6	3625.48	5700.03	36.40%	24/27	5700.03	0.00%	25/27	5699.51	0.01%
			0.8	4285.51	5861.51	26.89%	27/27	5861.51	0.00%	27/27	5861.51	0.00%
		avg				*34.23%*	*24.6/27*		*0.10%*	*26/27*		*0.00%*
	avg					*35.38%*	*25.1/27*		*0.05%*	*25.8/27*		*0.05%*
	Large	g_1	0.4	160.91	415.00	61.23%	14/24	415.00	0.00%	20/24	414.70	0.07%
			0.6	203.33	481.12	57.74%	19/24	480.70	0.09%	23/24	481.12	0.00%
			0.8	236.58	490.79	51.80%	24/24	490.79	0.00%	24/24	490.79	0.00%
		avg				*56.92%*	*19/24*		*0.03%*	*22.3/24*		*0.02%*
		g_2	0.4	8466.70	21165.20	60.00%	12/24	21077.83	0.41%	20/24	21165.20	0.00%
			0.6	10727.95	24327.12	55.90%	17/24	24301.91	0.10%	19/24	24327.12	0.00%
			0.8	12603.08	24814.95	49.21%	24/24	24814.95	0.00%	24/24	24814.95	0.00%
		avg				*55.04%*	*17.6/24*		*0.17%*	*21/24*		*0.00%*
	avg					*55.98%*	*18.3/24*		*0.10%*	*21.6/24*		*0.01%*
avg						*45.68%*	*21.7/24*		*0.08%*	*23.7/24*		*0.03%*

Table 3: Solution values for MASOP and HSOP

instances according to the number of customers, the kind of profits and the value of ω. Next, we report the average value of the initial solution, the average value of the best solution found by either MASOP or HSOP and the percentage improvement of the best solution over the initial solution. The average improvement is calculated as $\frac{z_{best}-z_{init}}{z_{best}}$ where z_{best} is the value of the best known solution and z_{init} is the value of the initial solution. Then, two blocks of three columns each follow related to MASOP and HSOP. They report, respectively: the number of times the corresponding algorithm found the best solution over the total number of instances in the class, the average solution value and the average gap with respect to the best known solution. The gap with respect to the best known solution is calculated as $\frac{z_{best}-z_H}{z_{best}}$ where z_H is the value of the solution provided by the algorithm considered.

From Table 3 we can see that, focusing on instances of Set 1, the effect of $MIPMove(T)$ is more evident on large instances than on small ones. In fact, while MASOP finds the best known value on 118 instances over 144, HSOP finds it on 89 instances only. The average gap has only a slight increase from MASOP to HSOP (from 0.10% to 0.12%) due to the fact that, on instances with profits g_1, HSOP behaves better than MASOP. On small instances instead, the two algorithms have very similar performances, with HSOP performing slightly better than MASOP. Concerning the comparison with the initial solution, the improvement produced by either MASOP or HSOP is remarkable, being 53.96% on average and 69.44% on large instances. Similar considerations can be made on instances of Set 2. Here, on large instances, MASOP finds the best solution on 123 instances over 144, while HSOP finds it on 100 instances. However, the difference in terms of average gap to best known value is larger than for instances of Set 1, being 0.01% for MASOP and 0.10% for HSOP. Considering the improvement with respect to the initial solution, we notice that it is lower than for instances of Set 1 but still 45.98% on average and 55.98% on large instances.

Finally, in Table 4 we report the computational times of MASOP and HSOP. The first three columns of the table are the same as in Table 3. The next two columns report the average computational times for the two algorithms.

Looking at Table 4, we can see that instances of Set 2 takes nearly the double of the time taken by instances of Set 1 for both algorithms. Also, MASOP takes nearly the triple of the time of HSOP on instances of Set 1 while less than the double on instances of Set 2. In any case, computing times of MASOP remain reasonable even for large instances.

4. Conclusions

In this paper we introduces the Set Orienteering Problem (SOP) which is a variant of the OP where customers are grouped in clusters and a profit is associated with each cluster. The profit is collected only if at least one vertex belonging to the cluster is visited. The objective is to find the tour that maximizes the collected profit and such that the corresponding duration does not exceed a given threshold. We propose a mathematical formulation and a matheuristic for the SOP. Computational tests on instances with up 1084 vertices show that the matheuristic produces high-quality results in reasonable computing times.

The SOP finds applications in the distribution of mass products where carriers stipulated contracts with customers organized in chains and the contracts are such that the carrier may choose which of the customer in the chain to visit in order to satisfy all the customers in the chain. Thus, this is a way to organize distribution processes which may provide advantages both to carriers, in terms of lower distribution costs, and for the customers, in terms of lower tariffs requested by the carriers.

References

[1] E. Angelelli, C. Archetti, M. Vindigni, The clustered orienteering problem, European Journal of Operational Research 238 (2014) 404–414.

[2] C. Archetti, M. Speranza, Arc routing problems with profits, in: A. Corberàn, G. Laporte (eds.), Arc Routing: Problems, Methods, and Applica-

	Size	p_g	ω	HSOP	MASOP
Set 1	Small	g_1	0.4	2.76	7.18
			0.6	3.57	8.52
			0.8	4.21	8.09
		avg		*3.51*	*7.93*
		g_2	0.4	2.86	7.39
			0.6	3.81	8.72
			0.8	4.59	7.96
		avg		*3.75*	*8.02*
	avg			*3.63*	*7.98*
	Large	g_1	0.4	37.57	172.46
			0.6	38.65	87.23
			0.8	36.06	76.95
		avg		*37.43*	*112.21*
		g_2	0.4	32.38	216.72
			0.6	33.35	93.88
			0.8	48.92	79.26
		avg		*38.22*	*129.95*
	avg			*37.82*	*121.08*
avg				*20.73*	*64.53*
Set 2	Small	g_1	0.4	3.51	6.15
			0.6	5.44	10.33
			0.8	7.01	13.57
		avg		*5.32*	*10.02*
		g_2	0.4	3.33	6.44
			0.6	5.42	10.29
			0.8	6.91	13.68
		avg		*5.22*	*10.14*
	Large	g_1	0.4	123.93	201.70
			0.6	115.96	140.16
			0.8	200.37	253.95
		avg		*146.75*	*198.60*
		g_2	0.4	130.84	271.93
			0.6	107.55	191.36
			0.8	226.59	287.12
		avg		*154.99*	*250.14*
	avg			*150.87*	*224.37*
avg				*79.08*	*121.16*

Table 4: Computing times for MASOP and HSOP

tions, MOS-SIAM Series on Optimization, SIAM, Philadelphia, 2014, pp. 257–284.

[3] C. Archetti, M. Speranza, D. Vigo, Vehicle routing problems with profits, in: P. Toth, D. Vigo (eds.), Vehicle Routing: Problems, Methods, and Applications, Second Edition, MOS-SIAM Series on Optimization 18, SIAM, Philadelphia, 2014, pp. 273–298.

[4] W. P. Coutinho, R. Q. do Nascimento, A. A. Pessoa, A. Subramanian, A branch-and-bound algorithm for the close-enough traveling salesman problem, INFORMS Journal on Computing 28 (4) (2016) 752–765.

[5] M. Fischetti, J. J. Salazar Gonzàlez, P. Toth, A branch-and-cut algorithm for the symmetric generalized traveling salesman problem, Operations Research 45 (3) (1997) 378–394.

[6] B. Gavish, S. Graves, The travelling salesman problem and related problems, Working paper gr-078-78, Operations Research Center, Massachusetts Institute of Technology (1978).

[7] A. Gunawan, H. C. Lau, P. Vansteenwegen, Orienteering problem: A survey of recent variants, solution approaches and applications, European Journal of Operational Research 255 (2) (2016) 315 – 332.

[8] B. Hu, G. R. Raidl, Effective neighborhood structures for the generalized traveling salesman problem, in: J. van Hemert, C. Cotta (eds.), Evolutionary Computation in Combinatorial Optimization: 8th European Conference, EvoCOP 2008, Naples, Italy, March 26-28, 2008. Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 36–47.

[9] G. Laporte, Y. Norbert, Generalized traveling salesman problem through n sets of nodes: An integer programming approach, INFOR 21 (1) (1983) 61–75.

[10] S. Lin, B. Kernighan, An effective heuristic algorithm for the traveling salesman problem, Operations Research 21 (1973) 498–516.

[11] T. Öncan, İ. K. Altınel, G. Laporte, A comparative analysis of several asymmetric traveling salesman problem formulations, Computers & Operations Research 36 (2009) 637–654.

[12] P. C. Pop, O. Matei, C. Sabo, A new approach for solving the generalized traveling salesman problem, in: M. J. Blesa, C. Blum, G. Raidl, A. Roli, M. Sampels (eds.), Hybrid Metaheuristics: 7th International Workshop, HM 2010, Vienna, Austria, October 1-2, 2010. Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 62–72.

[13] T. Tsiligirides, Heuristic methods applied to orienteering, Journal of the Operational Research Society 35 (1984) 797–809.