E. Angelelli, C. Archetti, M. Vindigni

The Clustered Orienteering Problem

WPDEM 2013/7
The Clustered Orienteering Problem

E. Angelelli  C. Archetti  M. Vindigni

Department of Economics and Management, University of Brescia, Italy, angele@eco.unibs.it, archetti@eco.unibs.it, vindigni@eco.unibs.it

Abstract

In this paper we study a generalization of the Orienteering Problem (OP) which we call the Clustered Orienteering Problem (COP). The OP, also known as the Selective Traveling Salesman Problem, is a problem where a set of potential customers is given and a profit is associated with the service of each customer. A single vehicle is available to serve the customers. The objective is to find the vehicle route that maximizes the total collected profit in such a way that the duration of the route does not exceed a given threshold. In the COP, customers are clustered in groups. A profit is associated with each group and is gained only if all customers belonging to the group are served. We propose two solution approaches for the COP: an exact and a heuristic one. The exact approach is a branch-and-cut while the heuristic approach is a tabu search. Computational results on a set of randomly generated instances are provided to show the efficiency and effectiveness of both approaches.

Keywords: Orienteering Problem, Branch-and-Cut, Tabu Search.

1 Introduction

The class of routing problems with profits is composed by a wide variety of problems which share the same characteristic: in contrast to what happens in the classical routing problems, not all customers need to be served. Instead, a profit is typically associated with each customer and the problem is to choose the right set of customers to serve satisfying a certain number of side constraints while optimizing a given objective function (maximize the total collected profit, minimize the traveling cost, maximize the difference among profits and costs,...). Among the routing problems with profits, the Traveling Salesman Problems with Profits (TSPPs) are problems where a single vehicle is available to carry out the service (see [3] for an excellent survey on TSPPs). In [3], TSPPs are classified in three main categories, depending on the objective function and side constraints: the Orienteering Problem (OP), also known as the Selective Traveling Salesman Problem, where the objective is to find the vehicle route that maximizes the total collected profit.
in such a way that the route duration does not exceed a given threshold; the Prize Collecting TSP (PCTSP),
which is the problem of finding the route that minimizes the traveling cost while ensuring that the profit
collected is greater than or equal to a minimum requested amount; finally, the Profitable Tour Problem (PTP)
which is the problem of finding the route that maximizes the difference between the total collected profit
and the traveling cost. The OP is certainly the variant that has received more attention in the literature. It
has been introduced in [14] and then studied in [7] as an application of the home fuel delivery problem. A
number of heuristic algorithm have been proposed (see [1], [5], [7], [8], [14] and [16]) and also efficient
exact algorithms (see [4], [6], [10] and [13]). The reader is refereed to [9] and [15] for excellent surveys on
the OP.

In this paper we address a generalization of the OP which we call the Clustered Orienteering Problem
(COP). In this problem, customers are clustered in groups. A profit is associated with each group and is
collected only if all customers in the group are served. Note that, if all groups are formed by a single
customers, the COP reduces to the OP.

The interest in studying the COP is motivated by the analysis of practical application problems that can
be formulated as variants or generalizations of the COP. Examples of such applications are mainly related
to the distribution of mass products, like in the case where customers are retailers belonging to different
supply chains and contracts are made between the carriers and the chains. Thus, if a carrier agrees to serve
a chain, he/she has to serve all retailers belonging to that chain. Another example is the case where products
are divided in brands; the carrier stipulates contracts with product shippers; retailers (customers) require a
certain amount of each product; in order to get the profit, the carrier has to serve all retailers requiring a
certain amount of product of the brand for which he/she has a contract. Also, a different case arises when
customers are clustered in areas and the profit is collected only if all customers in an area are served. This
happens for example in the case of companies providing waste collection services: they can be engaged by
municipalities to serve given areas and visiting all customers there.

The main contribution of the paper is the introduction and the study of the COP. We give a mathematical
formulation of the problem and propose two solution approaches: an exact solution approach which is a
branch-and-cut algorithm, and a heuristic algorithm based on a tabu search scheme. The exact solution
approach is able to solve instances with up to 318 vertices and 15 groups or 226 vertices and 25 groups
in one hour of computing time while, one the same classes of instances, the heuristic gives high quality
solutions in an extremely short amount of time. Three variants of the heuristic have been implemented and
tested also on larger instances which could not be solved by the exact algorithm. The variant based on a multi-start approach proved to be the best.

The paper is organized as follows. In Section 2 we describe the problem and propose a mathematical formulation. Section 3 is devoted to the branch-and-cut algorithm, together with the valid inequalities and branching rules we propose to enhance the efficiency of the approach. In Section 4 we describe the tabu search algorithm. In Section 5 we present the computational tests we made in order to verify the effectiveness of both the exact and the heuristic algorithm and we discuss the computational results. Conclusions are drawn in Section 6.

2 Problem description and formulation

The COP can be represented by an undirected graph $G = (V, E)$ where $V$ is the set of vertices and $E$ is the set of edges. Set $V = \{v_0, v_1, ..., v_n\}$ is formed by vertex $v_0$ which is the depot where the vehicle starts and ends its tour and vertices $v_1, ..., v_n$ which are the customers. A cover $S = \{S_1, S_2, ..., S_k\}$ of $V \setminus \{0\}$ is defined where $V \setminus \{0\} = \bigcup_{i=1}^{k} S_i$. In the following we call each element $S_i \in S$ a group. Each customer belongs to at least one group $S_i, i = 1, ..., k$. Note that a customer can belong to more than one group. An integer value $p_i$ is associated with each group $S_i$ and corresponds to the profit which is collected only if all customers in $S_i$ are served (visited) by the vehicle. A cost $t_e$ is related to each edge $e \in E$ and represents the time needed to traverse edge $e$. A single vehicle is available and a maximum time limit $T_{\text{max}}$ is imposed on the duration of the vehicle route. The objective is to find the route that maximizes the total collected profit and such that the duration is lower than or equal to $T_{\text{max}}$.

In order to give a mathematical formulation of the problem let us first introduce the following notation:

- $\delta(U)$: set of edges with one endpoint in $U$ and one endpoint in $V \setminus U \subseteq V$. For the ease of notation, we will write $\delta(j)$ for the set of edges adjacent to the single vertex $v_j$.
- $E(U)$: set of edges with both endpoints in $U \subseteq V$.
- $z_i$: binary variable equal to 1 if all customers in group $S_i \in S$ are served, 0 otherwise.
- $y_j$: binary variable equal to 1 if vertex $v_j \in V$ is served, 0 otherwise.
- $x_e$: binary variable equal to 1 if edge $e \in E$ is traversed, 0 otherwise.
The COP can then be formulated as follows:

\[
\text{max } \sum_{S_i \in S} p_i z_i \quad (1)
\]

\[
y_0 = 1 \quad (2)
\]

\[
\sum_{e \in \delta(j)} x_e = 2y_j \quad \forall v_j \in V \quad (3)
\]

\[
\sum_{e \in E} t_e x_e \leq T_{\text{max}} \quad (4)
\]

\[
\sum_{e \in E(U)} x_e \leq \sum_{v_j \in U \setminus \{v_i\}} y_j, \quad \forall U \subseteq V \setminus \{0\}, \forall v_i \in U \quad (5)
\]

\[
z_i \leq y_j, \quad \forall S_i \in S, \forall v_j \in S_i \quad (6)
\]

\[
z_i \in \{0, 1\} \quad \forall S_i \in S \quad (7)
\]

\[
x_e \in \{0, 1\} \quad \forall e \in E \quad (8)
\]

\[
y_j \in \{0, 1\} \quad \forall v_j \in V. \quad (9)
\]

The objective function (1) aims at maximizing the total collected profit. Constraint (2) imposes to visit the depot while (3) establishes to traverse two edges adjacent to each visited vertex. Inequality (4) imposes the maximum time limit on the route duration while (5) are the subtour elimination constraints. (6) imposes that all vertices belonging to a group must be served in order to get the corresponding profit. Finally, (7)–(9) are variable definitions.

In the following section we present a branch-and-cut algorithm for the exact solution of the COP.

### 3 A branch-and-cut algorithm

We implemented a branch-and-cut algorithm in order to solve model (1)–(9) which we call COP-CUT. Subtour elimination constraints (5) are originally removed from the formulation and inserted only once violated. In the following we describe the valid inequalities and the branching rules implemented in order to improve the efficiency of the algorithm, together with the separations algorithms which detect violated valid inequalities and subtour elimination constraints.


3.1 Valid inequalities

In order to strengthen the formulation, we introduced different valid inequalities. The first class of valid inequalities, which we call logical constraints, is the following:

\[ x_e \leq y_j \quad \forall v_j \in V, e \in \delta(j) \tag{10} \]

and establishes the relation between the \( x \) and the \( y \) variables.

A second class of inequalities, called connecting inequalities, states that, if a group is served, then there has to be at least two edges connecting the group with vertices outside the group:

\[ \sum_{e \in \delta(S_i)} x_e \geq 2z_i \quad \forall S_i \in S. \tag{11} \]

Moreover, we implemented a third class of valid inequalities, called clusters inequalities, which are based on the idea of identifying a cluster \( S' \subseteq S \) of groups which can not be feasibly visited altogether as this would violate the time constraint. The cluster inequalities are formulated as follows:

\[ \sum_{S_i \in S'} z_i \leq |S'| - 1 \quad \forall S' \subseteq S \text{ s.t. } TSP(S') > T_{\text{max}}. \tag{12} \]

Finally, the following two classes of valid inequalities are inserted each time a feasible solution or a new best solution is found, respectively.

Each time a new feasible solution is found, the following inequality is inserted:

\[ \sum_{S_i \notin C} z_i \geq 1 \tag{13} \]

where \( C \) is the set of groups visited in the feasible solution just found. The inequality establishes that at least one group not served in the current feasible solution solution has to be selected. Moreover, if the feasible solution just found improves the value of the best feasible solution, the following valid inequality is added:

\[ \sum_{S_i \in S} p_i z_i \geq \Psi \tag{14} \]

where
\[ \Psi = \min \left( \sum_{S \in C} p_i \mid C \subseteq S, \sum_{S \in C} p_i \geq p_{\text{best}} + 1 \right) \] (15)

and \( p_{\text{best}} \) is the value of the best feasible solution. The solution of (15) gives the minimal set of groups whose total profit, \( \Psi \), is greater than \( p_{\text{best}} \) and inequality (14) imposes to choose a set of groups such that the profit collected is greater than or equal to \( \Psi \). Note that lower bound \( \Psi \) may be strengthened by adding to problem (15) constraints that exclude clusters or superset of clusters which have been proved to be infeasible while separating inequalities (12).

### 3.2 Separation algorithms

Subtour elimination constraints (5) and valid inequalities (10), (11) and (12) are inserted only once violated, while (13) and (14) are inserted each time a new feasible or best solution is found, respectively. We look for the violation of valid inequalities (10), (11) and (12) up to the second level of the branch-and-bound tree. The order with which we insert cuts is the following: we first insert violated logical inequalities (10), followed by connecting inequalities (11), then subtour elimination constraints (5) and finally cluster inequalities (12). Every time we find at least one inequality which is violated, we insert the inequality and we return to the solution of the linear relaxation of the problem.

Logical and connecting inequalities are separated by simple enumeration. For the subtour elimination constraints we instead implemented the standard separation algorithm based on the solution of a maximum flow problem from the depot to each vertex of the auxiliary graph (see [12]).

Identifying violated inequalities (12) is a difficult task because of two main reasons: first of all, the number of inequalities is exponential in the number of groups and thus enumerating all of them is not a viable way. Second, once a cluster of groups is identified, in order to check if this cluster can be feasibly visited, it is necessary to solve a TSP on all vertices of the groups forming the cluster (plus the depot). It is thus of crucial importance to find a criterion to choose the cluster of groups on which a TSP will be solved, in order to avoid to solve many TSPs in vain. To this aim, we designed a procedure that identifies a cluster \( S' \) which violates constraints (12) and on which we successively solve the TSP. Let \( LB(z_i) \) and \( UB(z_i) \) be the lower bound and the upper bound on variable \( z_i \) defined by branching constraints, respectively. We define:
\[ C_1 = \{ S_i \in S | LB(z_i) = 1 \text{ in current node of the branch-and-bound tree} \} \]

\[ C_0 = \{ S_i \in S | UB(z_i) = 0 \text{ in current node of the branch-and-bound tree} \}. \]

The procedure we designed in order to identify violated constraints (12) consists in solving the following MILP problem:

\[
\begin{align*}
\max & \sum_{i=1}^{k} p_i \alpha_i \\
\sum_{i=1}^{k} \hat{z}_i \alpha_i & \geq \sum_{i=1}^{k} \alpha_i - 1 + \varepsilon \\
\sum_{i=1}^{k} \alpha_i & \geq 1 \\
\alpha_i = 1, & \forall S_i \in C_1 \\
\alpha_i = 0, & \forall S_i \in C_0 \\
\alpha_i & \in \{0, 1\}, \quad i = 1, \ldots, k
\end{align*}
\]

where \( \hat{z}_i \) corresponds to the value of \( z_i \) in the current optimal solution of the linear relaxation of (1)–(9) and \( \alpha_i \) is a binary variable equal to 1 if group \( i \) is inserted in cluster \( S' \). The model aims at finding the cluster of groups that violates constraint (12) while maximizing the total profit. Different objective functions can be thought of for the identification of a cluster of groups which violates inequality (12). For example, one may wish to find a cluster of groups which satisfies constraints (17)-(21) and maximizes the violation of inequality (12) or maximizes/minimizes the number of groups inserted in the cluster. We implemented different strategies and the following rule is the one with the best performance. We first identify the clusters of maximum and minimum cardinality violating constraints (12) by solving the following problems:

\[
\max \sum_{i=1}^{k} \alpha_i \quad \text{and} \quad \min \sum_{i=1}^{k} \alpha_i
\]
subject to constraints (17)-(21). Let us call $C_{\text{max}}$ the value of the optimal solution when the cardinality is maximized and $C_{\text{min}}$ the value of the optimal solution when the cardinality is minimized. We then solve problem (16)-(21) with the addition of the following constraint:

$$\sum_{S_i \in S} \alpha_i = \bar{c}.$$ 

Initially we set $\bar{c} = \lfloor \frac{C_{\text{max}} + C_{\text{min}}}{2} \rfloor$. Then, if the solution of the TSP on the cluster of groups identified by the optimal solution of (16)-(21) has a value greater than $T_{\text{max}}$, then the corresponding cut is added to the formulation. Otherwise, we set $C_{\text{min}} = \bar{c} + 1$, we redetermine the value of $\bar{c}$ as $\lfloor \frac{C_{\text{max}} + C_{\text{min}}}{2} \rfloor$ and we iterate. This rule for updating the value of $\bar{c}$ is due to the fact that, if we do not find a cluster of groups violating inequality (12) with a cardinality equal to the current value of $\bar{c}$, this is probably due to the fact that the value of $\bar{c}$ is too low, and thus we increase it. If instead we find a violated inequality, we insert it and we iterate.

To calculate the optimal TSP solution on the identified cluster of groups we use the Concorde library [2].

Separating inequalities (12) is thus time consuming as it involves solving a TSP for each cluster of groups identified by the solution of (16)-(21). In order not to solve many TSP in vain, each time problem (16)-(21) is solved we add a set of inequalities to its formulation which limit the search for the following solutions. In particular, let us call $S'$ the cluster of groups for which $\alpha_i = 1$ in the current solution of (16)-(21). Then, if $TSP(S') \leq T_{\text{max}}$, inequality (13) is added to (16)-(21) with $C = S'$ and $\alpha_i = z_i$.

The separation of inequalities (12) is stopped after 5 solutions of (16)-(21) without success, i.e., either (16)-(21) is infeasible or the set $S'$ identified by the solution is such that $TSP(S') \leq T_{\text{max}}$. The separation is stopped also when $h = C_{\text{max}}$ and (16)-(21) is solved without success.

Finally, in order to identify the value of $\Psi$ in inequality (14), the following MILP is solved:

$$\min \sum_{i=1}^{k} p_i \alpha_i$$

$$\sum_{i=1}^{k} p_i \alpha_i \geq p_{\text{best}} + 1$$

with the addition of all violated inequalities (12) found so far, where $\alpha_i = z_i$. 

8
3.3 Branching rules

As far as the separation algorithm of inequalities (12) is used, we implemented the following branching rule. If, in the current node of the branch-and-bound tree, while separating inequalities (12), we found a cluster \( C \) of groups for which \( \sum_{i \in C} z_i \geq |C| - 1 \) and \( TSP(C) \leq T_{\text{max}} \), then we generate two branches and set \( \sum_{i \in C} z_i \geq |C| \) on one branch and \( \sum_{i \in C} z_i \leq |C| - 1 \) on the other branch. The application of this branching rule is related to the separation of inequalities (12) and thus is made up to the second level on the branch-and-bound tree.

When the separation algorithm of inequalities (12) is not used, since the objective of the COP is to maximize the total collected profit and the profit is related to groups, we decided to give priority to the \( z \) variables when branching. In fact, the \( z \) variables are the only ones appearing in the objective function. The choice on which \( z \) variable to branch on is made on the basis of the default setting of the exact solver used.

4 A tabu search algorithm

As will be shown in Section 5.2, the COP-CUT algorithm is able to solve small to medium size instances. In order to solve larger instances, we propose a heuristic algorithm for the solution of the COP, in particular a tabu search algorithm which we call COP-TABU. The general scheme of COP-TABU is the following.

\textbf{COP-TABU}

\begin{align*}
\text{COMPUTE AN INITIAL SOLUTION } s_0 \\
& \ s^* \leftarrow s_0 \\
& \ s \leftarrow s_0 \\
\text{While A STOPPING CRITERION IS NOT MET do} \\
& \ \text{GENERATE THE NEIGHBORHOOD } N(s) \\
& \ \text{CHOOSE THE BEST SOLUTION } s' \in N(s) \\
& \ s \leftarrow s' \\
& \ \text{If } s \text{ IS BETTER THAN } s^* \text{ then} \\
& \ \quad s^* \leftarrow s \\
\text{End If}
\end{align*}
UPDATE THE TABU LIST

UPDATE THE LONG-TERM MEMORY

End While

RETURN $s^*$

In the following we explain in detail each step of COP-TABU. Procedures UPDATE THE TABU LIST and UPDATE THE LONG-TERM MEMORY are explained before procedure CHOOSE THE BEST SOLUTION $s' \in N(s)$ as they are needed to understand how we choose $s' \in N(s)$.

COMPUTE AN INITIAL SOLUTION $s_0$

The initial solution is generated by first ordering all groups randomly and then inserting them sequentially in $s_0$, if the corresponding solution is feasible. The procedure stops when all groups have been considered for insertion.

GENERATE THE NEIGHBORHOOD $N(s)$

Given the current solution $s$, let $C(s)$ be the set of groups visited in solution $s$ and $\bar{C}(s)$ be its complement. The neighborhood $N(s)$ is made by the following moves:

- Given $S_i \in \bar{C}(s)$, insert $S_i$ in $s$ if the corresponding solution is feasible.
- Given $S_i \in C(s)$, remove $S_i$ from $s$.

Then insertion of a cluster $S_i$ in a solution $s$ is made by solving the TSP on all vertices belonging to $C(s) \cup S_i$ with the Lin-Kernigham algorithm [11]. The removal of a cluster always leads to a feasible solution thus it does not require any TSP calculation.

UPDATE THE TABU LIST

Every time a group is inserted (removed) from $s$, then it is tabu to remove (insert) it from $s$ for $\alpha$ iterations.

UPDATE THE LONG-TERM MEMORY
Let $\eta_i$ be the number of iterations group $S_i$ has remained in the current solution. $\eta_i$ is set to one each time $S_i$ is inserted in $C(s)$. At each iteration, $\eta_i$ is set to 0 if $S_i$ is removed from the current solution, otherwise it is increased by one.

**Choose the best solution $s' \in N(s)$**

In order to avoid to uselessly explore the entire neighborhood, we defined a rule which split it into different neighborhood sets. Each neighborhood set is identified according to the move applied on the current solution $s$ and to the fact that the tabu status is taken into account or not.

1. **Non-Tabu Insertion.** Insert in $s$ a non-tabu group in $\tilde{C}(s)$, if any.
2. **Old Removal.** Remove from $s$ a group $S_j \in C(s)$ for which $\eta_j > \beta$, if any.
3. **Non-Tabu Removal.** Remove from $s$ a non-tabu group $S_j \in C(s)$, if any.
4. **Tabu Insertion.** Insert in $s$ a tabu group $S_i \in \tilde{C}(s)$ whose insertion leads to a solution with a higher value than the aspiration level of $S_i$, if any. The aspiration level of group $S_i$ is defined as the solution value obtained the last time $S_i$ was in $C(s)$.
5. **Random Removal.** Choose randomly a group in $C(s)$ and remove it from $s$.
6. **Random Insertion.** Choose randomly a group in $\tilde{C}(s)$ and insert it in $s$.

Note that a move is applied only if it leads to a feasible solution. Also, the best move is implemented. Thus, in order to avoid to calculate many TSP and then choose the best move, when an insertion move is considered (either tabu or non-tabu), first groups are ordered on the basis of a non-increasing profit. The first group in the list which can be feasibly inserted in $s$ is then considered. Similarly, for the non-tabu removal, we first order the non-tabu groups on the basis of a non-decreasing profit and we remove the first in the list.

Given a neighborhood set $\tilde{s}$, a move is admissible for $\tilde{s}$ if it it belongs to the set of moves defining neighborhood set $\tilde{s}$ and if the corresponding solution is feasible. Neighborhood sets are explored according the previous order and the first admissible move is implemented, as this rule leads to the choice of the best move. This implies that, if there is at least one admissible move for Non-Tabu Insertion, then the best one is implemented. Otherwise, it means that no group in $\tilde{C}(s)$ can be inserted, either because it is tabu or because its insertion leads to an infeasible solution. In this case, we remove a group from $C(s)$ with Old Removal.
which checks if there are groups which have remained in \( C(s) \) for a high number of iterations (more than \( \beta \)) and remove the oldest one. Note that, in this case, the group removed is not tabu as many iterations have elapsed from its last move. If there is no group \( S_i \in C(s) \) with \( \eta_i \geq \beta \), then we apply *Non-Tabu Removal* and remove the non-tabu group in \( C(s) \) with the lowest profit. The reason why we decided to remove first the ‘old’ groups is that typically they correspond to groups with a high profit and thus they would never be removed when considering the ordering on the basis on a non-decreasing profit. If no group is removed, it means that all groups \( S_i \in C(s) \) are tabu and have \( \eta_i < \beta \). In this case *Tabu Insertion* is applied. *Random Removal* is applied only when no tabu group in \( \bar{C}(s) \) beats its aspiration level while *Random Insertion* is applied only when solution \( s \) is empty.

A final note has to be made on the TSP calculations. Each time a TSP is calculated on a set \( \tilde{C} \) of groups, the information concerning feasibility of set \( \tilde{C} \) is stored in memory. Then, if set \( \tilde{C} \) is feasible, i.e., \( TSP(\tilde{C}) \leq T_{\text{max}} \), COP-TABU will not calculate the TSP on \( \tilde{C} \) and on each set \( \tilde{C}' \subseteq \tilde{C} \) in the following iterations. If instead \( TSP(\tilde{C}) > T_{\text{max}} \), then COP-TABU will not calculate the TSP on \( \tilde{C} \) and on each set \( \tilde{C}' \supseteq \tilde{C} \) in the following iterations.

Note that in COP-TABU the value of \( TSP(C) \) is calculated through the Lin-Kernigham algorithm and thus it is a heuristic value. This means that COP-TABU can discard feasible solutions.

We implemented three variants of COP-TABU:

- **Basic**: COP-TABU is stopped after \( \Gamma \) iterations.
- **Multi-start**: COP-TABU is restarted \( \gamma \) times from \( \gamma \) different random initial solutions. To have a fair comparison with the basic COP-TABU each run is stopped after \( \frac{\Gamma}{\gamma} \) iterations.
- **Reactive**: COP-TABU is stopped after \( \Gamma \) iterations. Moreover, each time we find a new best solution \( s^* \), \( \alpha \) is decreased by 25% of its current value. On the contrary, when the current solution does not improve \( s^* \), the value of \( \alpha \) is increased by 2. In any case, \( \alpha \) is always kept in the interval \([1, k-1]\), where \( k \) is the number of groups. The reason behind this rule for updating the value of \( \alpha \) is that, when a new best solution is found, we want to deeply explore the solution space around this solution thus we need to reduce drastically the value of \( \alpha \) to avoid to miss good solutions because they are classified as tabu. On the other side, when the best solution is not improved, we want to avoid to come back to the current solution and thus we increase the value of \( \alpha \). However, in order to avoid to restrict too much the set of non-tabu solutions, \( \alpha \) is increased gradually by a constant value equal to 2.
5 Computational tests

In this section we describe the computational tests we made in order to verify the performance of both the COP-CUT and the COP-TABU algorithm. In Section 5.1 we describe how we generated the instances while Section 5.2 is dedicated to computational results.

5.1 Test instances

To the best of our knowledge, COP has never been studied previously in the literature. Thus, there are no benchmark instances and we had to generate them. To this aim, we take TSP benchmark instances from the TSPLIB95 library available at the following url: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95. We note that, in the instances of the TSPLIB95 library, vertices are numbered from 1. In our tests, the first vertex coincides with \( v_0 \), the depot.

We take all instances with a number of vertices ranging from 42 to 532 which are 57 in total. We keep the data concerning the location of the vertices and we generate the remaining data as follows:

1. Groups: the number \( k \) of groups has been set to the following values: 10, 15, 20, 25. Groups are generated by considering the vertices sequentially with respect to their number and inserting them in each group in such a way to obtain groups of similar size and which are partially overlapping.

2. \( p_i \): in order to generate the profit of each group \( S_i \), we first assigned a profit to each vertex, excluding the depot. The profit of a group is then given as the sum of the profits of the vertices in the group. The profits are generated according to two rules as done in [4] for the OP. The first rule set the profit of each vertex equal to 1. The second rule set the profit of a vertex \( j \) equal to \( 1 + (7141j + 73) mod(100) \) in order to obtain pseudo-random profits.

3. \( T_{\text{max}} \): we set the value of \( T_{\text{max}} \) as \( \theta \cdot TSP^* \) where \( TSP^* \) is the optimal value of the TSP over all vertices.

   We considered three values of \( \theta \): 1/4, 1/2, and 3/4.

Thus, we generated in total \( 57 \times 4 \times 3 \times 2 = 1368 \) instances. The instances can be found at the following url: http://www-c.eco.unibs.it/~angele/COP.zip.
5.2 Computational results

In this section we present the computational results of the tests we made in order to verify the efficiency of both COP-CUT and COP-TABU. We first analyze the performance of COP-CUT in Section 5.2.1 and then we focus on COP-TABU in Section 5.2.2. All tests have been made on an Intel Xeon W3680 six-core CPU 3.33GHz, Windows 7 Professional-64 bit operating system with 12Gb ram. COP-CUT and the basic branch-and-cut algorithm, described in the next section, are implemented in Concert Technology with CPLEX 12.2. Computing times are expressed in seconds.

5.2.1 Performance of COP-CUT

COP-CUT has been tested on instances with up to 318 vertices as larger instances could not be solved within the time limit which was set to 1 hour. In order to verify the performance of the algorithm, we compare it against the solution of formulation (1)-(9). The aim of our tests is to verify the efficiency of valid inequalities (12), (13), (14) and of the branching rule described in Section 3.3. As inequalities (10) and (11) were already proven to be effective in [4], we did not focus our tests on their efficiency and we inserted them in the algorithm that solves the basic formulation. In the following, we call BASIC this branch-and-cut algorithm, i.e., the one based on formulation (1)-(9) plus inequalities (10) and (11). A further notice on the BASIC branch-and-cut is that we set a priority on the \( z \) variables when branching. In fact, when using the default CPLEX parameter, we got extremely poor results.

Results are shown for instances with a number of vertices ranging from 200 to 318, for a total of 288 instances. For instances with a lower number of vertices COP-CUT does not compare favorably with the BASIC algorithm. In fact, while all instances are solved by the BASIC algorithm, 145 of them are not solved by COP-CUT. We thus decided to present the results for instances with at least 200 vertices where we can see the advantages of using COP-CUT instead of the BASIC algorithm.

Figures 1-4 report the results related to the optimality gap at the end of the computation. The optimality gap is calculated as \( \frac{\bar{z} - \tilde{z}}{\tilde{z}} \) where \( \bar{z} \) is the upper bound at the end of the computation and \( \tilde{z} \) is the value of the best feasible solution found by the algorithm. We report the results classified by number of vertices (Figure 1), number of groups (Figure 2), kind of generation of profits (Figure 3) and value of \( \theta \) (Figure 4). This in order to detect which feature of the problem influences its complexity the most. In Figure 3, ‘g1’ stands for the instances where the profit of each vertex is set to 1 while ‘g2’ is the class of instances with random profits.
In Figure 4, ‘q1’ stands for the instances where the $\theta = 1/4$, ‘q2’ is the class of instances where $\theta = 1/2$ while in ‘q3’ $\theta = 3/4$.

![Optimality gap with respect to the number of vertices](image)

Figure 1: Optimality gap with respect to the number of vertices

If we first focus on detecting which characteristic influences the complexity of the problem the most, we can see that there is no clear evidence from the figures. As far as the number of vertices, the instances with 200 and 262 vertices seem to be the most difficult ones while the behavior of both algorithms is quite fluctuating. Concerning the number of groups, there seems to be no influence on the complexity of the problem as the average optimality gap is quite stable (Figure 2). The class of instances ‘g2’ is more difficult than the class ‘g1’. As far as the value of $\theta$, the results should be evaluated carefully. In fact, for some instances with $\theta = 1/4$, no group can be feasibly served and thus the optimal solution value is 0. In this case, if the algorithm is not able to prove the optimality within the computing time, then the optimality gap is 100%. This explain the high value of the average optimality gap for the ‘q1’ instances. If we instead compare ‘q2’ and ‘q3’, we see that ‘q2’ is the most difficult class. This is due to the fact that the value of $\theta$ is more binding in ‘q2’ than in ‘q3’. In fact, while in ‘q3’ the value of $\theta$ is 3/4 and thus a small number of groups are not visited in the optimal solution, in ‘q2’ $\theta = 1/2$ and nearly half of the groups are not visited. Thus, in ‘q2’ deciding which groups have to be visited or not is more complex than in ‘q3’.

If we now concentrate on the performance of the algorithms, figures 1-4 shows that COP-CUT performs better than the BASIC algorithm. With the exception of instances with 200 and 226 vertices, the average optimality gap of COP-CUT is on average lower than the one of the BASIC algorithm. If we look at the
Figure 2: Optimality gap with respect to the number of sets

Figure 3: Optimality gap with respect to the kind of generation of profits
number of sets or the kind of generation of profits, the average gap of COP-CUT is always lower than the one of the BASIC algorithm. Finally, if we concentrate on the value of $\theta$, COP-CUT performs better for classes ‘q2’ and ‘q3’. As mentioned earlier, the results on ‘q1’ are distorted by the null value instances.

We perform the same analysis as before also on the computational time. The results are shown in Figures 5-8.

The figures show that on average COP-CUT requires a lower computational time, especially for class ‘g1’ of instances, 10 groups and $\theta = 3/4$. As expected, Figure 8 shows that the class of instances which requires more computing time is the one with $\theta = 1/2$.

A final observation is that the number of instances solved to optimality is 87 for the COP-CUT algorithm and 98 for the BASIC algorithm. This have to be weighted up with the results related to the optimality gap and solution time illustrated previously. The separation of inequalities (12) is time consuming and this penalize COP-CUT especially in the solution of smaller instances. However, these inequalities are fundamental to reduce the optimality gap when the size of the instances increases.

### 5.2.2 Performance of COP-TABU

We now present the computational tests we made to verify the performance of COP-TABU. The three variants of the algorithm have been tested on all instances. For instances with up to 318 vertices we compared the
Figure 5: Computational time with respect to the number of vertices

Figure 6: Computational time with respect to the number of sets
Figure 7: Computational time with respect to the kind of generation of profits

Figure 8: Computational time with respect to the value of $\theta$
solution with the optimal solution given by COP-CUT or the \textit{BASIC} algorithm, if available, or with the best upper bound. Note that the number of these instances is much higher than the one considered in the previous section as we now include also the instances with less than 200 vertices. In particular, we compare COP-TABU with the best upper bound and the best feasible solution found by COP-CUT and the \textit{BASIC} algorithm. In the following, we will use the term ‘branch-and-cut algorithm’ to indicate the best between COP-CUT and the \textit{BASIC} algorithm. For larger instances, as branch-and-cut is not able to solve them, we make a comparison between the three variants of COP-TABU. The behavior of COP-TABU is strictly related to the number of groups, whereas there seems to be no evident relation with the number of vertices, kind of generation of profits and value of $\theta$. Thus, we will present the results on the basis of the number of groups. Preliminary tests showed that the following parameter values give the best results: $\Gamma = 1000$, $\beta = \frac{F}{10}$, $\gamma = 3$, $\alpha = 0.3|S|$. Thus, we use the previous parameter values in the tests which are shown in the following.

Figures 9-13 refer to the instances with up to 318 vertices for which we compare the three variants of COP-TABU with the results given by the branch-and-cut. In particular, Figure 9 reports the average gap with respect to the upper bound while in Figure 10 we focus only on instances which are solved to optimality by the branch-and-cut. In Figure 11 we compare the solution given by COP-TABU with the best feasible solution found by the branch-and-cut. Figure 12 refers to the iteration at which the best solution (which coincides with the final solution) has been found. For COP-TABU \textit{Multi-start}, this number is cumulative over the three restartings. Finally, Figure 13 reports the average computing time.

![Figure 9: Error with respect to the upper bound](image-url)
Figure 10: Error with respect to the optimal solution

Figure 11: Error with respect to the best feasible solution found by the branch-and-cut
Figure 12: Iteration at which the best solution has been found

Figure 13: Computational time
Looking at Figure 9 we can notice that the average gap with respect to the upper bound is below 7.5% for COP-TABU Reactive and COP-TABU Basic while it is below 7% for COP-TABU Multi-start. Detailed results show that this error go up to 100% for all variants of COP-TABU. This happens on instances with $\theta = \frac{1}{4}$ where COP-TABU fails to find a feasible solution when it exists. This is due to the fact that the TSP is solved heuristically. For the other classes of instances, the error go up to 57.16% and this happens for all the three variants of COP-TABU on the same set of instances. However, when comparing the solution given by COP-TABU with the best feasible solution found by the branch-and-cut, we found that, for the instances where the error with respect to the upper bound is so high, COP-TABU finds a solution which is 24.24% better than the one found by the branch-and-cut. This induces us to conclude that these big errors are mostly due to the poorness of the upper bound.

For instances solved to optimality by the branch-and-cut (Figure 10), the error is always below 1.8% for COP-TABU Multi-start and slightly higher than 2.5% for COP-TABU Reactive and COP-TABU Basic. More in detail, the maximum error of COP-TABU with respect to the optimal solution is 43.75% for the Reactive and Basic variants and 37.5% for the Multi-start variants. These high errors occurs for those instances for which the solution space is very narrow (typically the ones with $\theta = 1/4$) and either the algorithm finds the optimal solution or the second best is very far from the optimum. Moreover, the number of instances solved to optimality by the branch-and-cut is 994. Of those, COP-TABU Multi-start finds 833 optimal solutions while COP-TABU Basic and COP-TABU Reactive find both 797 optimal solutions. This allows us to conclude that all versions of COP-TABU are extremely effective and COP-TABU Multi-start has the best performance. This is confirmed by Figure 11 which shows that COP-TABU is extremely efficient in providing high quality solutions: in fact, COP-TABU finds on average better solutions than the branch-and-cut on all classes of instances, and the improvement increases with the number of groups. A further consideration which highlights the efficiency of COP-TABU is that computational time is reasonable as shown in Figure 13. The same figure also shows that the computational time is strictly related to the number of groups. This is expected as the higher the number of groups, the more the number of moves that have to be evaluated by COP-TABU. Finally, Figure 12 shows that on average COP-TABU finds the best solution after less than 180 iterations. Given that COP-TABU stops after 1000 iterations, this means that we could remarkably reduce the computing time while assuring a high quality of solutions. We decided to maintain a stopping criterion of 1000 iterations as the computational time is still reasonable and we got slightly worse solutions on a subset of instances when reducing this value.
If we now compare the three versions of COP-TABU, we can see that COP-TABU *Multi-start* beats both COP-TABU *Basic* and COP-TABU *Reactive* in terms of solution quality. From the computational time point of view, COP-TABU *Basic* is the fastest algorithm while COP-TABU *Multi-start* is on average the slowest. Moreover, COP-TABU *Multi-start* requires a higher number of iterations to find the best solution.

For instances with more than 318 vertices we could not compare COP-TABU with the branch-and-cut, thus we decided to compare the three variants of COP-TABU. The results are summarized in Figure 14 which reports the average error with respect to the best solution found by the three algorithms. We do not report figures concerning the average number of iterations needed to find the best solution and the average solution time as they show a similar behavior as the one illustrated in figures 12 and 13. We simply note that the average number of iterations needed to find the best solution exceeds 180 for all versions of COP-TABU on instances with 25 groups but is anyway lower than 200. The average computational time increases when the number of groups is higher, as shown in Figure 13, and reaches 950 seconds for the COP-TABU *Multi-start* on instances with 25 groups. For the same number of groups, the computational time is 860 seconds for COP-TABU *Reactive* and 840 seconds for COP-TABU *Basic*.

![Figure 14: Error with respect to the best solution found](image)

The results confirm that COP-TABU *Multi-start* is the best heuristic in terms of solution quality requiring a solution time which is slightly higher than the other two algorithms. A final observation has to be made with respect to the computing time. The stopping criterion is the same for all versions of COP-TABU, i.e., 1000 iterations (999 for COP-TABU *Multi-start*). The difference in terms of computing time is due to the
number of times the Lin-Kernigham algorithm is called. As mentioned before, each time a TSP is calculated, COP-TABU stores the information about the cluster of groups on which the calculation has been made in order to avoid to repeat the calculation in the following iterations. Thus, a shorter computational time means that the algorithm has visited a higher number of identical solutions and thus is less effective, as proved by the results.

6 Conclusions

In this paper we analyze a new variant of the Orienteering Problem, the Clustered Orienteering Problem, where customers are clustered in groups and a profit is associated with each group and is collected only if all vertices of the group are served. This problem comes from the analysis of practical applications in supply chain management where products of specific brands have to be distributed to all customers belonging to the same supply chain.

We present a mathematical formulation together with different valid inequalities and embed them in a branch-and-cut algorithm which is able to solve instances with up to 318 vertices in one hour of computing time. Computational results show that the number of vertices is not the main characteristic of the problem that influences its complexity. In fact, the problem complexity depends also on other features such as how binding the maximum time constraint is. We notice that the complexity of the problem does not seem to depend on the number of groups and this is quite surprising.

To solve larger instances, we develop a heuristic algorithm, in particular a tabu search algorithm. The main feature of this algorithm is its simplicity: the neighborhood is based on the simple addition and removal of a group. Despite its simplicity, the results prove that it can give high quality solutions in a very short computing time.

Future research could be focused on the extension of the COP to the case of multiple vehicles considering also additional constraints like vehicle capacity or time windows. Both the solution algorithms presented in this paper could be adapted to deal with these extensions.
References


