

Hybrid Heuristic for the Clustered Orienteering Problem

Ala-Eddine Yahiaoui, Aziz Moukrim and Mehdi Serairi

Sorbonne universités, Université de technologie de Compiègne, CNRS
Heudiasyc UMR 7253, CS 60 319, 60 203 Compiègne cedex
{ala-eddine.yahiaoui, aziz.moukrim, mehdi.serairi}@hds.utc.fr

Abstract. This paper addresses the Clustered Orienteering Problem, a recent variant of the Orienteering Problem. In this variant, customers are grouped into subsets called *clusters*. A profit is assigned to each cluster and is collected only if all customers belonging to the cluster are served. The objective is to visit the customers of a subset of clusters in order to maximize the total collected profit with respect to a travel time limit. Our solution method is based on the *order first-cluster second* approach. It incorporates a *split* procedure that converts a *giant tour* into an optimal solution. Experiments conducted on benchmark instances show that our algorithm outperforms the existing methods in the literature. Actually, we have found the best known solution for 916 instances from 924 with strict improvement of 82 instances.

Keywords: Clustered Orienteering Problem · Adaptive Large Neighborhood Search · Split · Branch and Bound · Knapsack Problem

1 Introduction

The Orienteering Problem (OP) is a well studied variant of the Traveling Salesman Problem (TSP). In the OP, a profit is associated with every customer to represent the value of service, and the aim is to select a subset of customers to visit in order to maximize the total collected profit without exceeding a predefined travel time limit.

Recently, a new generalization of the OP was introduced by Angelelli et al. [1] called the Clustered Orienteering Problem (COP). In this problem, customers are grouped into subsets called *clusters*. Unlike the OP, profits are associated with the clusters instead of the customers. The profit of a given cluster is gained only if all of its customers are served.

A COP instance, that we denote by I_{COP} , is modeled as a complete undirected graph $G = (V, E)$ where $V = \{1, \dots, n\} \cup \{0\}$ is the set of vertices representing customers and the depot, and E is the set of edges. A cost $c(e)$ is assigned to each edge $e \in E$ which represents the travel time needed to cross e . We assume that travel times satisfy the triangle inequality. A cover $S = \{S_1, S_2, \dots, S_K\}$ is a set of K clusters where $\cup_{i=1}^K S_i = V \setminus \{0\}$. Each customer can belong to more than one cluster. A profit P_i is associated with each cluster S_i which is collected

only if customers belonging to cluster S_i are all served. One vehicle is available to serve customers with a maximum travel time T_{max} . It is noteworthy to mention that there is no requirement on the order of visits, i.e. a vehicle can alternate the visits between customers belonging to different clusters.

Fig.1 shows a feasible COP solution for an instance where $|S| = 3$, $|V| = 9$. Customers with two circles means that they belong to two clusters. In this solution, only the customers of cluster S_2 and cluster S_3 are served.

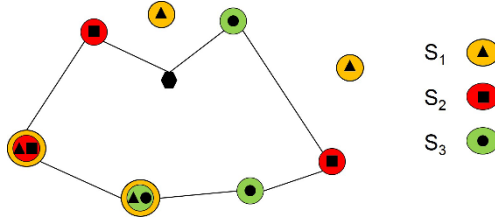


Fig. 1: Example of COP solution

The interest in the COP arises in many real-life applications that can be modeled as variants or generalizations [1]. One of the applications of the COP is when customers are grouped into clusters according to their geographical locations, and a profit is gained only if all customers belonging to a particular area are served. Another example is in the distribution of mass products, where customers are supply chains that contain many retailers. In the case where a contract is made between a carrier and a supply chain, the carrier should serve all the retailers of that supply chain.

Angelelli et al. [1] proposed an exact and a heuristic method to solve the COP. The exact method is a branch and cut algorithm based on the OP formulation proposed in Fischetti et al. [4]. Angelelli et al. [1] solved a linear relaxation of the model without subtour elimination constraints. These constraints are added to the model once violated. The branch and cut algorithm is able to solve optimally small and medium-sized instances. To tackle large-scale instances, Angelelli et al. [1] proposed a heuristic method based on tabu search (TS). TS used an ordered set of insertion and removal moves. Each time a cluster is inserted, TS used a TSP heuristic called Lin-Kernighan heuristic [5] to check the move feasibility.

In this paper, we propose a hybrid heuristic scheme based on the *order first-cluster second* approach [8] to solve the COP. The first component is a meta-heuristic scheme called Adaptive Large Neighborhood Search (ALNS) heuristic, whose aim is to generate *giant tours* with good quality. The *giant tours* are then provided to the second component which is a split procedure in order to extract solutions with better profit. The split is based on a branch and bound algorithm that incorporates a knapsack-based upper bound to fathom inferior nodes.

The remainder of this paper is as follows. The global scheme of the proposed heuristic is introduced in Section 2. The ALNS heuristic is detailed in Section 3.

Our split algorithm is presented in Section 4. Computational results are presented in Section 5. Finally, we conclude by some remarks in Section 6.

2 Heuristic global scheme

In the last decade, numerous heuristics based on the *order first-cluster second* approach have been proposed for the VRP and its variants [8]. This approach consists of two phases: the ordering phase in which a giant tour covering all customers is constructed. In the second phase, a split procedure is used to extract the optimal solution while respecting the predefined order of customers. The first split method was introduced by Beasley in [2] for the CVRP. Then, this method was incorporated within a genetic algorithm by Prins in [7].

For selective VRP, in most cases it is impossible to serve all the customers due to the travel time limit. Thus, the objective of a split procedure is to select a subset of customers that satisfies the objective function. Vidal et al. [10] and Vargas et al. [9] studied some selective problems like the Team Orienteering Problem, Capacitated Profitable Tour Problem, Covering Tour Problem, etc. while considering the giant tour. Vidal et al. [10] modeled the problem as a resource constrained shortest path. To solve the problem, they proposed an efficient split procedure based on dynamic programming in order to maximize the total collected profit. Vargas et al. [9] used also in their heuristic a dynamic programming based split to minimize the total travel time. For more detailed literature on the *order first-cluster second* approach, we refer the reader to [8].

Our solution method adopts also the *order first-cluster second* approach. Algorithm 1 describes the global scheme of our heuristic. It is composed of two main components: an ALNS metaheuristic and a split procedure. The ALNS generates solutions with good quality in a short time (line 4). From a given solution, a giant tour is constructed by randomly inserting the unrouted customers (line 5). Then, the giant tour is given to the split procedure in order to extract a solution with better profit (line 6). We use in Algorithm 1 $Eval(X)$ to denote the profit of a solution X . This process is iterated until a stop condition is reached. In our algorithm, we consider two conditions: the first one is the maximum number of iterations which is fixed at n , where n is the number of customers. The second stop condition is the maximum number of iterations without improvement, which is fixed at the average number of customers per cluster.

3 Adaptive large neighborhood search

The main feature of the ALNS is the use of multiple neighborhoods in parallel during the search process [6]. These different neighborhoods are identified by a set of competing removal and insertion operator. An operator is defined as a fast heuristic that explores a large part of the neighborhood in a polynomial time. In each iteration, the algorithm selects a removal and an insertion operator based

Algorithm 1: GLOBAL SCHEME

Input: *Solution* X **Output:** *Solution* X_{best}

```

1  $X_{best} \leftarrow X$ 
2  $LB \leftarrow Eval(X)$ 
3 repeat
4    $ALNS(X)$ (see Section 3)
5   Construct a giant tour  $GT$  from  $X$ 
6    $X \leftarrow SPLIT(GT, LB)$ (see Section 4)
7   if ( $Eval(X) > Eval(X_{best})$ ) then
8      $X_{best} \leftarrow X$ 
9      $LB \leftarrow Eval(X)$ 
10 until (stop condition is reached)
11 return  $X_{best}$ 

```

on statistics gathered during the search process. This characteristic improves the flexibility of the heuristic to tackle a wide variety of instances.

Our ALNS scheme includes one removal operator and a set of three insertion operators. We use a local search operator called 2-opt to improve the travel time of the current solution. This operator is called at each iteration between the removal and the insertion operator.

Random removal operator

This operator selects a random number of clusters between 1 and d_{max} and removes their customers from the current solution. Note that customers which are shared with other clusters in the solution are not removed. The worst-case complexity of this operator is $O(n * K)$.

The parameter d_{max} is a diversification/intensification parameter. If it is small, the heuristic tries to intensify the search in a limited neighborhood. On the other hand, if d_{max} is large, it helps the heuristic to modify a large part of the solution in order to escape from local optima. In our heuristic, d_{max} is set to initial value equal to 3, then it is increased by 1 after each iteration without improvement. Note that d_{max} must not exceed the current number of routed clusters. Once the current solution is improved, d_{max} is set to 3.

Insertion operators

Insertion operators are incorporated in a global scheme that inserts unrouted clusters one by one in the current solution. A cluster is unrouted if and only if at least one of its customers is unrouted. At each iteration, an unrouted cluster is randomly selected, then its unrouted customers are identified (probably some of its customers have been already inserted) and given to one of the insertion operators. The process is iterated until either no further insertions are possible or all the clusters are inserted.

Best insertion operator (BIO): This operator evaluates all feasible insertions for each unrouted customer. Then the best insertion with the smallest travel time gap is selected. The process is iterated until either all customers are inserted or the solution cannot accept other customers. The complexity of this operator is $O(n^3)$.

Insertion with regret Operator (IRO): IRO evaluates all feasible insertions for each unrouted customer. Then, it calculates the gap in terms of travel time between the two best insertions of each customer. We call this gap as *regret*. Then it selects the customer with the highest *regret* and inserts it in the solution. The process is iterated until either all customers are inserted or no customer can be added to the solution. The complexity of this operator is $O(n^3)$.

Random Best Insertion Operator (RBIO): RBIO randomly selects one unrouted customer then evaluates all of its feasible insertions that respect the travel time limit. The best insertion is then selected. The process is iterated until either all customers are inserted or no customer can be added to the solution. The complexity of this operator is $O(n^2)$.

Adaptive weight adjustment

An important aspect of the ALNS is the dynamic weight adjustment carried out during the search process. Weights are associated with insertion operators and are initialized using the same value. Then, these weights are dynamically changed during the search progress according to the performance of each operator. The aim is to give larger weights to operators which have contributed better to the solution process. The criteria used to measure how much an operator contributes during the search process is based on the quality of the solution found after each iteration:

- if it is a new best solution, it gives a large weight to the operator.
- if it is better than the current solution, it gives a medium weight to the operator.
- if it is worse than the current solution, it gives a small weight.

For more details about the update procedure, the reader is referred to Pisinger and Ropke [6].

4 Split procedure

We propose in the following a split procedure based on a branch and bound scheme. The aim of the split is to find the subset of clusters that maximizes the collected profit while respecting the order of customers in π and the travel time limit. Before detailing our split procedure, let us first introduce a preliminary result. This result is used afterwards in the upper bound.

Algorithm 2: ALNS

Input: *Solution* X **Output:** *Solution* X_{best}

```

1  $d_{max} \leftarrow 1 + \text{rand}()\%3$ 
2  $X_{best} \leftarrow X$ 
3 repeat
4   Remove  $d_{max}$  clusters from  $X$ 
5   Apply 2-opt on  $X$ 
6   Select an insertion operator  $i$ 
7   Apply  $i$  on  $X$ 
8   if ( $\text{Eval}(X) > \text{Eval}(X_{best})$ ) then
9      $X_{best} \leftarrow X$ 
10     $d_{max} \leftarrow 1 + \text{rand}()\%3$ 
11  else  $d_{max} \leftarrow d_{max} + 1$ 
12
13  Update weights using the adaptive weight adjustment procedure
14 until (stop condition is reached)
15 return  $X_{best}$ 

```

4.1 Preliminary result

In this subsection, we present a relaxation scheme for the COP based on the OP.

Definition 1. *Given a COP instance I_{COP} . We define an OP associated instance I_{OP} composed of the same set of vertices $V = \{0, 1, \dots, n\}$ and the same set of edges E . Profits of customers in I_{OP} are computed as follows:*
 $\rho_j = \sum_{i:j \in S_i} \frac{P_i}{|S_i|}$. *In fact, the ratio $\frac{P_i}{|S_i|}$ could be interpreted as the contribution of the customer j to the cluster S_i . Finally, the maximal travel time is T_{max} .*

Proposition 1. *For any COP instance I_{COP} , the optimal objective value of the associated OP instance, I_{OP} , represents an upper bound on the profit of I_{COP} .*

Proof. We prove in the following that the optimal solution of a given I_{COP} is a feasible solution for I_{OP} with a profit lower than or equal to the optimal objective value of the I_{OP} .

Assume that S^* is the set of clusters of the optimal solution of I_{COP} with a total collected profit $P_{cop}(S^*) = \sum_{i:S_i \in S^*} P_i = P_{cop}^*(I_{COP})$. Let V^* be the set of customers belonging to S^* . It is obvious that this optimal solution is a feasible solution for the I_{OP} and its profit is $P_{op}(V^*) = \sum_{j \in V^*} \rho_j$. We denote

by $P_{op}^*(I_{OP})$ the optimal objective value for I_{OP} .

$$\begin{aligned}
P_{op}(V^*) &= \sum_{j \in V^*} \rho_j = \sum_{j \in V^*} \sum_{i: j \in S_i} \frac{P_i}{|S_i|} \\
&= \sum_{j \in V^*} \sum_{i: j \in S_i \text{ and } S_i \in S^*} \frac{P_i}{|S_i|} + \sum_{j \in V^*} \sum_{i: j \in S_i \text{ and } S_i \notin S^*} \frac{P_i}{|S_i|} \\
&= P_{cop}(S^*) + \sum_{j \in V^*} \sum_{i: j \in S_i \text{ and } S_i \notin S^*} \frac{P_i}{|S_i|} \\
&= P_{cop}^*(I_{COP}) + \sum_{j \in V^*} \sum_{i: j \in S_i \text{ and } S_i \notin S^*} \frac{P_i}{|S_i|} \tag{1}
\end{aligned}$$

We conclude that an optimal solution for I_{COP} is feasible for the I_{OP} . Furthermore, $P_{cop}^*(I_{COP}) = P_{cop}(S^*) \leq P_{op}(V^*) \leq P_{op}^*(I_{OP})$. \square

Let us consider now a giant tour $\pi = (\pi_1, \pi_2, \dots, \pi_n)$ that covers all the customers of I_{COP} . The giant tour π imposes an order of visit among all the customers of I_{COP} . This can be seen as a derived instance I'_{COP} , in which arcs that do not respect this ordering are not considered. The following corollary holds.

Corollary 1. *Given a COP instance I_{COP} , its associated instance I_{OP} and a giant tour π . The optimal objective value of I_{OP} while considering π represents an upper bound on the optimal objective value of I_{COP} w.r.t. to π .*

4.2 Principle of the split

The goal is to calculate a partial sequence σ that visits the customers of a subset of clusters in order to maximize the total collected profit while preserving the original order of customers in π . To that end, the branch and bound algorithm explores a search tree generated according to decisions made on clusters.

In the root node, an arbitrary order of branching is established among clusters. In each node of the search tree, the possible decision that can be made regarding a given cluster is whether it is selected or rejected. This leads to a binary search tree with at most $2^{K+1} - 1$ nodes.

Several components are embedded within the branch and bound algorithm in order to achieve high performance. These components include in addition to the branching scheme, a suitable node selection strategy, an upper bound to fathom inferior nodes, a feasibility test to discard unfeasible nodes. In what follows, we describe the different components implemented in our branch and bound algorithm.

Before proceeding further, we distinguish in each node η three subsets of clusters: the selected clusters denoted by S_s^η , the removed clusters denoted by S_r^η and the potential clusters denoted by S_p^η representing the remainder set of clusters on which decision has not been made yet.

4.3 Knapsack-based upper bound

Vargas et al. [9] proposed a dynamic programming split procedure that incorporates a lower bound based on the Fractional Knapsack Problem (FKSP). We propose in this paper an upper bound that is also based on the FKSP. We make use of the cluster constraint in order to improve this upper bound.

Given a giant tour π and a node η in the branch and bound tree. We consider the Knapsack instance I_{FKSP} in which we associate an item to each potential customer. A customer is considered as potential if it belongs at least to one of the potential clusters S_p^η and does not belong to any of the selected clusters S_s^η .

The profit of a given item/potential customer π_j is calculated using Definition 1. Note that to calculate these profits in a node η , we consider only contributions related to potential clusters S_p^η and we discard those related to removed clusters S_r^η . Consequently, in a given node η and for a given potential customer π_j , we have: $\rho_{\pi_j}^\eta = \sum_{i: \pi_j \in S_i \text{ and } S_i \in S_p^\eta} \frac{P_i}{|S_i^\eta|}$, where $|S_i^\eta|$ is the number of potential customers belonging to cluster S_i in the node η .

The weight $w_{\pi_j}^\eta$ of the item/potential customer π_j in a given node η is modeled by the minimal insertion cost. Assume that I_j^η is the set of all valid insertion positions composed of a predecessor and a successor of π_j in π , i.e. $I_j^\eta = \{(\pi_l, \pi_r) | l < j < r, \pi_l, \pi_r \in S_s^\eta \cup S_p^\eta\}$. Thus the minimal insertion cost is calculated as $w_{\pi_j}^\eta = \min\{c(\pi_l, \pi_j) + c(\pi_j, \pi_r) - c(\pi_l, \pi_r) | (\pi_l, \pi_r) \in I_j^\eta\}$, where $c(\pi_l, \pi_r)$ is the travel time between customers π_l and π_r .

To model the knapsack size W^η , we proceed as follows. We consider the partial sequence that contains the customers of the selected clusters S_s^η . Assume D is the travel time needed to go from the depot, visit all these customers and return back to the depot. W^η is modeled as the residual distance, i.e. $W^\eta = T_{max} - D$.

Proposition 2. *Given a giant tour π and a node η in the branch and bound tree, the optimal objective value of the I_{FKSP} previously defined represents an upper bound on the profit of the I_{COP} while considering π and η .*

Proof. Given a giant tour π covering all the customers of I_{COP} and a node η . We construct $FKSP$ instance I_{FKSP} in which, each item/ potential customer π_j has a weight w_j^η and a profit $\rho_{\pi_j}^\eta$. According to Corollary 1, an upper bound on I_{OP} is also an upper bound on the I_{COP} while considering π and η . In the following, we prove that the optimal solution of I_{FKSP} is an upper bound on I_{OP} while considering π and η .

Assume σ^η is the optimal partial sequence in the node η and $\delta^\eta(\pi_j)$ is the insertion cost of the customer π_j in σ^η . According to the definition of the minimal cost insertion, we observe that $w_j^\eta \leq \delta^\eta(\pi_j)$ for any potential customer π_j in S_p^η . Consequently, the optimal solution for the I_{FKSP} is an upper bound on the profit of I_{OP} while considering π and η . \square

Each customer can have n^2 possible insertion positions. In the following, we propose to reduce this number. Assume that π_j is a potential customer and $(\pi_l, \pi_r) \in I_j^\eta$ is a possible insertion position. This couple of customers must satisfy the following rules.

- The first rule is that (π_l, π_r) must not skip any visited customer, i.e. (π_l, π_r) is considered only if:

$$\bar{A}j' / (l < j' < j \text{ or } j < j' < r) \quad \text{and} \quad \pi_{j'} \in S_s^\eta \quad (2)$$

- The second rule is that for any skipped customer, its cluster set must not include the cluster set of any of the involved customers in the insertion $(\pi_l, \pi_r \text{ or } \pi_j)$. Let us define $\Omega(i)$ as the set of clusters which customer i is included in, i.e. (π_l, π_r) is considered only if:

$$\bar{A}j' / (l < j' < j \text{ or } j < j' < r) \\ \text{and} \quad (\Omega(\pi_l) \subseteq \Omega(\pi_{j'}) \text{ or } \Omega(\pi_r) \subseteq \Omega(\pi_{j'}) \text{ or } \Omega(\pi_j) \subseteq \Omega(\pi_{j'})) \quad (3)$$

For computational efficiency, the best insertion for each customer is pre-computed beforehand and saved. Each time a cluster is selected or rejected, this list of possible insertions is updated.

4.4 Feasibility check

Feasibility check (*FC*) is done every time a potential cluster is selected. This is done by computing the length of the partial sequence that contains only the customers of the selected clusters while considering the given order of the giant tour. If the length of this partial sequence exceeds T_{max} , and due to the triangle inequality, node η can be pruned. The complexity of this test is $O(n)$.

4.5 Local search procedure

We propose to improve the split procedure by integrating a Local Search heuristic (*LS*). The LS uses some relevant information from the enumeration tree in order to explore efficiently the search space alongside with the branch and bound. The solution value obtained by LS is used also as a lower bound in the branch and bound.

Each time the LS is called in a given node η , it considers only the selected and the potential sets of clusters $S_s^\eta \cup S_p^\eta$. The LS consists of two phases: a destruction phase which is used as a perturbation technique. It removes a small number of clusters from the current solution. This number is chosen randomly between 1 and 3.

The second phase is a constructive heuristic which tries to insert clusters one by one until either the solution cannot accept additional clusters or there is no clusters left. It randomly selects in each iteration one unrouted cluster and tries to insert its customers in the current solution. To check the feasibility of a cluster insertion, this procedure calls an Iterative Destructive Constructive Heuristic (IDCH) proposed in [3]. If IDCH fails to insert the customers, the Lin-Kernighan TSP heuristic [5] is used (see Algorithm 3).

Algorithm 3: ITERATIVE INSERTION

Input: Solution X **Output:** Solution X

```

1  $\Delta \leftarrow$  unrouted clusters of  $X$ 
2  $insert \leftarrow true$ 
3 while ( $\Delta \neq \emptyset$  and  $insert = true$ ) do
4    $insert \leftarrow false$ 
5   foreach ( $k \in \Delta$ ) do
6     if ( $IDCH(X, k) = true$ ) then
7        $\Delta \leftarrow \Delta \setminus \{k\}$ 
8        $insert \leftarrow true$ 
9       break
10    else if ( $LinKernighan(X, k) = true$ ) then
11       $\Delta \leftarrow \Delta \setminus \{k\}$ 
12       $insert \leftarrow true$ 
13      break
14     $\Delta \leftarrow \Delta \setminus \{k\}$ 
15 return  $X$ 

```

4.6 Beam search

When the number of clusters becomes large, computational time dramatically increases. To cope with this problem, we propose to limit the number of nodes generated during the search process. The main idea is to explore the search tree using a breath-first search (BFS) and impose a limit on the number of nodes expanded in each level of the tree. Consequently, this scheme does not guarantee that the solution found is optimal. It is important to select in each level the most promising nodes to be expanded, so that a good-quality solution could be found. To this end, we use the knapsack upper bound described in Section (4.3) as a selection criteria. Another important aspect is the number of nodes selected at each level. This parameter was fixed after experimentation at K nodes per level.

Algorithm 4 describes the whole split procedure. We use in Algorithm 4 two ordered lists, one is the active list, and the second is temporary. The lower bound LB is initialized by the best objective value obtained by the global heuristic.

5 Computational results

Our heuristic is coded in C++ using the Standard Template Library (STL) for data structures. Experiments were conducted on a computer with Intel Xeon X7542 CPU@2.66 GHz and a Linux OS 64 bits.

In order to verify the efficiency of our approach, we used benchmark instances designed in [1]. The benchmark is derived from 57 instances of TSPLIB with the number of vertices ranging from 42 to 532. For each base instance of TSPLIB, a

Algorithm 4: SPLIT

Input: giant tour GT , Lower bound LB
Output: best solution X_{best}
Data: Ordered lists of size K : $actList$, $tmpList$

- 1 **Initialization:** ordered list of the clusters $Order$ used as branching strategy, current level $L \leftarrow 0$, current node $e \leftarrow 0$, $actList \leftarrow e$, $tmpList \leftarrow \emptyset$
- 2 **while** ($actList \neq \emptyset$ and $L < K$) **do**
- 3 Select the best node e in $actList$ based on Knapsack UB (See Section 4.3)
- 4 Expand e to two nodes e_1 and e_2 based on $Order(L)$ (See Section 4.2)
- 5 **foreach** ($e \in \{e_1, e_2\}$) **do**
- 6 **if** (e is infeasible) **then continue** (See Section 4.4)
- 7 **if** ($Knapsack\ UB\ of\ (e) \leq LB$) **then continue** (See Section 4.3)
- 8 $tmpList \leftarrow tmpList \cup \{e\}$
- 9 Extract solution X from e
- 10 Apply Local Search on X (See Section 4.5)
- 11 **if** ($Eval(X) > Eval(X_{best})$) **then**
- 12 $X_{best} \leftarrow X$
- 13 **if** ($Eval(X) > LB$) **then** $LB \leftarrow Eval(X)$
- 14 **if** ($actList = \emptyset$) **then**
- 15 $actList \leftarrow tmpList$
- 16 $tmpList \leftarrow \emptyset$
- 17 $L++$
- 18 Select the best node e in $actList$ based on Knapsack UB (See Section 4.3)
- 19 Extract solution X from e
- 20 **if** ($Eval(X) > Eval(X_{best})$) **then** $X_{best} \leftarrow X$
- 21 **return** X_{best}

set of derived instances for the COP is constructed according to different values assigned to the following parameters:

1. Number of clusters: It varies between the values 10, 15, 20 and 25.
2. Profits of clusters: Two models are used, the first is deterministic while the second is random.
3. T_{max} : Given TSP^* the optimal value of TSP over all vertices of the base instance, T_{max} is set to the values $\frac{1}{2}TSP^*$ and $\frac{3}{4}TSP^*$.

As a result, 16 different instances are derived from each TSP instance. Furthermore, 12 other instances are added to the biggest class with 532 vertices. These instances have a larger number of clusters (50, 75 and 100). Thus, the total number of instances is 924. The instances can be found at the following URL: <http://or-brescia.unibs.it/>. For detailed description of instance generation, the reader can refer to Angelelli et al. [1].

5.1 Parameter setting

The execution of the LS procedure inside the branch and bound algorithm seems to be expensive in terms of computational time. In order to reach the best performance of our algorithm in terms of solution quality and computational time, we propose to tune the number of calls of the LS procedure inside the branch and bound algorithm. We call this parameter N_{LS} . In our experiments, N_{LS} takes different values of $k \times C_{avg}$ where ($k = 1, 10, 20, 30, 40$). C_{avg} represents the average number of customers per cluster. We carried out these experiments on a representative sample composed of 22 instances. These instances are chosen between the most difficult ones for which high values of N_{LS} are needed to obtain solutions with good quality.

To measure the performance of each configuration, we used the relative gap to the best solution found in the literature, denoted by RPE and the average CPU time. To calculate the RPE , we recorded the Best Known Solution for each instance (Z_{best}), and also we recorded the maximal score (Z_{max}) realized by our heuristic. The relative percentage error RPE of a given instance using (4).

$$RPE = \frac{Z_{best} - Z_{max}}{Z_{best}} \times 100 \quad (4)$$

According to Fig. 2, the value $20 \times C_{avg}$ gives the best compromise in terms of RPE and CPU time. In fact, the RPE tends to stabilize at a value near to zero when the N_{LS} exceeds $20 \times C_{avg}$, whereas the CPU time continues to increase. As a result, we set N_{LS} at $20 \times C_{avg}$.

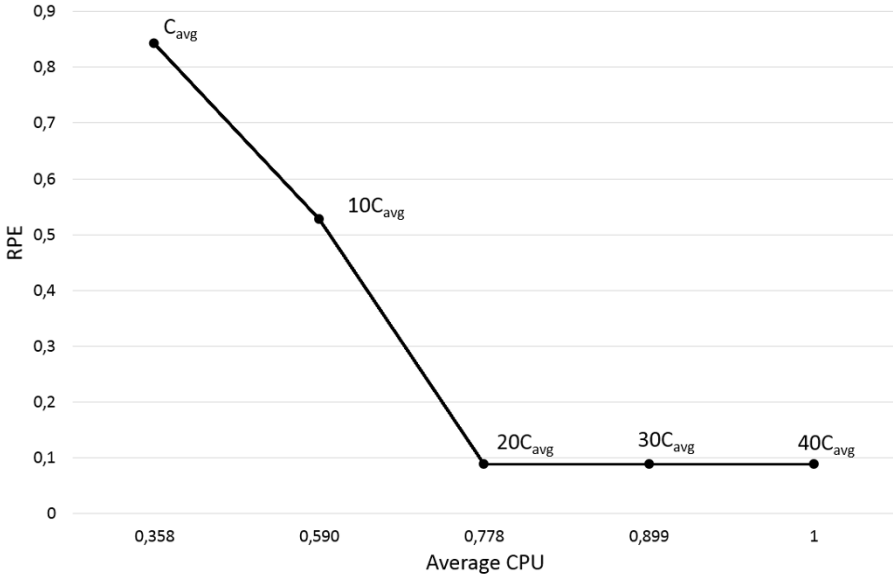


Fig. 2: Performance of our heuristic with different values of N_{LS}

5.2 Performance comparison

Results achieved by our algorithm are compared to the different versions of tabu heuristics proposed in [1]. Three versions of tabu were presented: COP-TABU-Basic, COP-TABU-Multistart and COP-TABU-Reactive.

Table 1. shows the results organized per class of instances. As described earlier, each class is composed of 16 instances, except the last one (*att532*), which is composed of 28 instances. We run our algorithm 10 times per instance as in [1]. For each method, we provide the number of instances per class for which the Best Known Solution was found (*BKS*). We report also the relative percentage error per class (*CRPE*) which is the average *RPE* per class of instances. The average CPU time (*CPU*) for each class is compared to the best results of each heuristic.

The results show clearly that our algorithm outperforms existing methods in the literature. It succeeds to reduce the *CRPE* to less than 0.011 against 1.498 for COP-TABU-Basic, 0.841 for COP-TABU-Multistart and 0.327 for COP-TABU-Reactive. Our heuristic found up to 916 *BKS* against 656, 720 and 816 for the three tabu versions. Furthermore, new *BKS* were found for 82 instances. In terms of CPU time, our heuristic consumes lower computational time than the three tabu versions. In fact, our heuristic has an average CPU time of 136.33s, against 153.71s for COP-TABU-Basic, 174.37s for COP-TABU-Multistart and 223.88s for COP-TABU-Reactive.

Table 1: PERFORMANCE OF OUR HEURISTIC

Class	COP-TABU-Basic			COP-TABU-Multistart			COP-TABU-Reactive			Our Contribution		
	<i>BKS</i>	<i>CRPE</i>	<i>CPU</i>	<i>BKS</i>	<i>CRPE</i>	<i>CPU</i>	<i>BKS</i>	<i>CRPE</i>	<i>CPU</i>	<i>BKS</i>	<i>CRPE</i>	<i>CPU</i>
<i>dantzig42</i>	16	0	13.27	16	0	17.77	16	0	38.95	16	0	0.55
<i>swiss42</i>	13	0.719	15.38	14	0.281	23.09	15	0.013	31.93	16	0	0.56
<i>att48</i>	16	0	18.24	16	0	26.08	15	0.062	38.76	16	0	1.51
<i>gr48</i>	11	5.709	13.74	12	3.184	26.02	16	0	37.96	16	0	0.78
<i>hk48</i>	15	1.250	20.58	16	0	30.76	15	0.315	37.68	16	0	1.05
<i>eil51</i>	11	2.181	15.92	11	2.181	24.46	15	0.242	36.83	14	0.228	1.11
<i>berlin52</i>	15	0.548	38.88	15	0.120	53.39	15	0.120	60.41	16	0	2.42
<i>brazil58</i>	13	0.573	58.99	14	0.115	75.72	16	0	83.97	16	0	4.51
<i>st70</i>	11	1.303	23.18	11	1.012	38.95	12	0.639	48.01	16	0	2.44
<i>eil76</i>	9	6.407	24.50	10	4.050	33.74	15	0.125	45.84	16	0	2.41
<i>pr76</i>	11	1.014	21.40	13	0.105	30.88	15	0.009	54.76	16	0	4.79
<i>gr96</i>	12	0.612	44.07	13	0.116	51.35	14	0.025	68.19	16	0	5.49
<i>rat99</i>	12	1.752	32.99	12	0.127	52.03	15	0.034	63.65	15	0.079	7.03
<i>kroA100</i>	11	6.013	44.65	14	0.123	50.98	14	0.429	52.62	16	0	3.92
<i>kroB100</i>	15	0.714	47.96	16	0	58.94	16	0	62.20	16	0	3.87
<i>kroC100</i>	10	3.687	37.55	15	0.269	48.74	14	0.452	59.42	16	0	3.73
<i>kroD100</i>	10	1.879	36.85	11	1.247	56.70	13	0.520	69.57	16	0	4.94
<i>kroE100</i>	12	2.889	46.59	12	1.374	48.83	14	0.270	62.77	16	0	3.69
<i>rd100</i>	12	1.431	36.51	13	1.030	47.81	15	0.568	82.29	16	0	4.89
<i>eil101</i>	7	2.495	32.97	12	0.729	44.62	16	0	79	16	0	5.87
<i>lin105</i>	11	1.393	36.06	13	0.461	52.48	14	0.348	105.21	16	0	11.42
<i>pr107</i>	13	6.350	72.19	15	0.203	86.35	15	0.160	135.39	16	0	36.10
<i>gr120</i>	10	2.917	50.87	11	2.856	66.36	14	0.185	105.25	16	0	10.43
<i>pr124</i>	14	1.180	80.33	16	0	88.26	16	0	150.15	16	0	18.7
<i>bier127</i>	12	0.873	63.05	14	0.108	94.57	15	0.005	149.64	16	0	14.65
<i>ch130</i>	7	4.016	49.79	9	2.949	64.58	12	1.376	106.57	16	0	10.91
<i>pr136</i>	12	1.588	59.86	14	0.949	71.37	15	0.694	121.5	16	0	14.83
<i>gr137</i>	15	0.156	82.07	16	0	104.45	16	0	181.54	16	0	14.25
<i>pr144</i>	16	0	168.25	16	0	175.28	16	0	247.29	16	0	29.07

continued on next page

Table 1 – continued from previous page

Class	COP-TABU-Basic			COP-TABU-Multistart			COP-TABU-Reactive			Our Contribution		
	<i>BKS</i>	<i>CRPE</i>	<i>CPU</i>	<i>BKS</i>	<i>CRPE</i>	<i>CPU</i>	<i>BKS</i>	<i>CRPE</i>	<i>CPU</i>	<i>BKS</i>	<i>CRPE</i>	<i>CPU</i>
<i>ch150</i>	8	2.684	34.19	8	2.543	53.97	14	0.554	101.37	16	0	13.87
<i>kroA150</i>	9	1.002	36.84	13	0.228	50.6	14	0.074	102.11	16	0	13.13
<i>kroB150</i>	8	2.456	40.06	10	2.127	56.69	14	0.621	107.93	16	0	12.31
<i>pr152</i>	15	0.545	120.08	16	0	164.81	16	0	248.1	16	0	26.30
<i>u159</i>	6	3.300	113.36	9	2.373	125.51	8	1.447	184.68	16	0	48.45
<i>si175</i>	16	0	47.69	16	0	63.36	16	0	126.80	16	0	227.71
<i>brg180</i>	12	0.656	54.29	13	0.578	72.18	15	0.091	127.74	16	0	166.41
<i>rat195</i>	12	0.531	68.18	10	0.209	78.52	14	0.401	172	16	0	48.04
<i>d198</i>	15	0.062	172.56	16	0	217.29	16	0	368.98	16	0	40.15
<i>kroA200</i>	11	1.130	55.09	12	1.093	76.17	14	1.052	139.43	15	0.035	30.45
<i>kroB200</i>	8	2.610	71.45	10	1.978	87.73	13	0.129	142.43	16	0	29.4
<i>gr202</i>	11	1.256	88.17	12	1.001	121.27	16	0	236.24	16	0	56.19
<i>ts225</i>	12	0.259	162.94	12	0.158	189.13	15	0.019	234.81	16	0	65.60
<i>tsp225</i>	9	1.583	87.78	9	0.495	102.99	11	0.142	180.26	16	0	64.19
<i>pr226</i>	12	0.872	244.84	12	0.787	268.33	15	0.042	331.10	16	0	84.61
<i>gr229</i>	15	0.023	109.99	15	0.023	121.07	15	0.023	170.85	16	0	34.08
<i>gil262</i>	7	8.107	57.09	6	4.296	84.20	10	2.441	135.48	15	0.032	70.85
<i>pr264</i>	11	4.230	151.96	10	4.243	208.51	14	0.323	304.70	16	0	100.79
<i>a280</i>	11	0.159	99.98	12	0.156	150.54	10	0.255	191.39	15	0.003	249.43
<i>pr299</i>	10	1.097	105.14	10	1.089	125.98	12	0.614	205.23	16	0	221.36
<i>lin318</i>	8	1.009	247.01	9	0.870	260.81	11	0.492	311.25	16	0	309.03
<i>rd400</i>	10	2.014	100.44	11	1.435	147.13	12	1.413	203.08	16	0	322.72
<i>fl417</i>	11	1.055	518.97	12	0.397	577.53	13	0.079	708.57	16	0	362.65
<i>gr431</i>	12	0.788	236.75	15	0.009	252.35	16	0	280.53	16	0	251.45
<i>pr439</i>	11	0.685	180.16	13	0.074	221.23	14	0.058	324.17	16	0	316.19
<i>pcb442</i>	11	0.390	151.28	11	0.610	199.82	13	0.593	274.18	16	0	499.62
<i>d493</i>	7	1.157	418.35	9	1.208	419.66	12	1.051	515.84	16	0	638.77
<i>att532</i>	16	3.218	3815.65	19	2.684	3927.68	24	1.769	4082.2	26	0.275	3343.6
<i>Total</i>	656	1.498	153.719	720	0.841	174.37	816	0.327	223.88	916	0.011	136.338

6 Conclusion and future work

In this paper, we proposed a hybrid heuristic for the Clustered Orienteering Problem. This heuristic is composed of a split procedure that evaluates efficiently *giant tours* and an Adaptive Large Neighborhood Search heuristic. The split procedure is based on a branch and bound scheme, in which an efficient upper bound based on the Knapsack Problem is used. A Local Search procedure is also incorporated inside the split procedure. The LS is applied each time on a subset of clusters in order to find better combination of clusters quickly. The computational results show clearly the efficiency of our method compared to the existing heuristic methods. Many improvements have been achieved as well as new Best Known Solutions.

As future work, our aim is to propose different extensions for the COP, including the case of multiple vehicles. Also, additional constraints like time windows or vehicle capacity should be considered.

Acknowledgements

The authors would like to thank the Hauts-de-France region and the European Regional Development Fund (ERDF) 2014/2020 for the funding of this work.

This work was carried out in the framework of ANR project TCDU (Collaborative Transportation in Urban Distribution ANR-14-CE22-0017) and of Labex MS2T funded through the program "Investments for the Future" managed by the National Agency for Research (Reference ANR-11-IDEX-0004-02).

References

1. Enrico Angelelli, Claudia Archetti, and Michele Vindigni. The clustered orienteering problem. *European Journal of Operational Research*, 238(2):404–414, 2014.
2. John E Beasley. Route first-cluster second methods for vehicle routing. *Omega*, 11(4):403–408, 1983.
3. Hermann Bouly, Duc-Cuong Dang, and Aziz Moukrim. A memetic algorithm for the team orienteering problem. *JOR*, 8(1):49–70, 2010.
4. Matteo Fischetti, Juan Jose Salazar Gonzalez, and Paolo Toth. Solving the orienteering problem through branch-and-cut. *INFORMS Journal on Computing*, 10(2):133–148, 1998.
5. Shen Lin and Brian Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.
6. David Pisinger and Stefan Ropke. Large neighborhood search. In *Handbook of metaheuristics*, pages 399–419. Springer, 2010.
7. Christian Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12):1985–2002, 2004.
8. Christian Prins, Philippe Lacomme, and Caroline Prodhon. Order-first split-second methods for vehicle routing problems: A review. *Transportation Research Part C: Emerging Technologies*, 40:179–200, 2014.
9. Leticia Vargas, Nicolas Jozefowicz, and Sandra Ulrich Ngueveu. A dynamic programming operator for tour location problems applied to the covering tour problem. *Journal of Heuristics*, 23(1):53–80, 2017.
10. Thibaut Vidal, Nelson Maculan, Luiz Satoru Ochi, and Puca Huachi Vaz Penna. Large neighborhoods with implicit customer selection for vehicle routing problems with profits. *Transportation Science*, 50(2):720–734, 2015.

Computational Logistics

8th International Conference, ICCL 2017, Southampton,
UK, October 18-20, 2017, Proceedings

Bektas, T.; Coniglio, S.; Martinez-Sykora, A.; Voß, S.
(Eds.)

2017, XIV, 588 p. 129 illus., Softcover

ISBN: 978-3-319-68495-6