The clustered team orienteering problem

Ala-Eddine Yahiaoui, Aziz Moukrim, Mehdi Serairi

The Clustered Team Orienteering Problem

Ala-Eddine Yahiaoui[a], Aziz Moukrim[a], Mehdi Serairi[a]

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

Abstract

In this paper, we present a new variant of the Clustered Orienteering Problem (COP) that we refer to as the Clustered Team Orienteering Problem (CluTOP). In this problem, customers are grouped into subsets called clusters. A profit is associated with each cluster and is gained only if all of its customers are served. A set of available vehicles with a limited travel time collaborates in order to visit the customers of the clusters. The objective is to maximize the total collected profit with respect to a travel time limit.

The first contribution of this paper consists of an exact method based on a cutting planes approach. This method includes the consideration of a set of valid inequalities. In particular an incompatibility-cluster-based valid inequality is proposed. Moreover a pre-processing procedure is considered in order to compute the incompatibilities between clusters. The second contribution is a hybrid heuristic that combines an Adaptive Large Neighborhood Search (ALNS) and an effective split procedure. These two components cooperate together for the purpose of exploring both direct representation and giant tours search spaces.

Experimental results show that the cutting plane based algorithm outperforms the state-of-the-art exact methods, in the particular case of a single vehicle by solving 61 additional instances. Moreover, the hybrid heuristic succeeds in finding 38 new best known solutions for the case of one vehicle. For the case with multiple vehicles, new benchmark instances are generated based on those introduced for the COP. Regarding the performance of the methods, the heuristic method finds the optimal solution for almost all the instances solved by the exact methods.

Keywords: team orienteering problem · cluster · cutting plane · adaptive large neighborhood search · Split procedure

1. Introduction

In this paper, we propose a new variant of the vehicle routing problems with profits, which we refer to as the Clustered Team Orienteering Problem (CluTOP). In this variant, customers are grouped into subsets called *clusters*. A profit is assigned to each cluster, which is gained only if all of the customers

in the cluster are visited. A set of identical vehicles cooperates in order to maximize the total collected profit w.r.t. the limited travel time imposed on each vehicle.

A special case of the CluTOP is when all the clusters are formed by single customers. This problem is known as the Team Orienteering Problem (TOP), one of the most studied routing problems with profits in the literature [3]. The TOP is inspired from the sport game of orienteering, in which a set of players of the same team work together in order to collect as many rewards as possible from a set of locations w.r.t. a time limit imposed for each player [5]. Many exact and heuristic methods have been proposed for the problem. The reader can refer to [3], [9], [12] and [19] for surveys on variants, applications and solution methods as well.

The CluTOP is also a generalization of the Clustered Orienteering Problem (COP) proposed in Angelelli et al. [1]. In this problem, a single vehicle is used to serve the selected clusters. The authors in [1] proposed two approaches to solve the COP. The first approach is based on a branch-and-cut algorithm. They proposed two branching schemes and suitable valid inequalities in order to enhance the solution process. The second approach consists of a Tabu Search heuristic. Several insertion and removal operators were proposed. Three versions of this heuristic were introduced and compared.

The concept of cluster has been used in several variants of vehicle routing problems to denote a subset of customers. However, the signification of this concept differs depending on the problem. In the Generalized Traveling Salesman Problem (GTSP) [11], for example, the customers are grouped into subsets called *clusters*, and the salesman has to visit at least one customer from each *cluster*. In the Clustered Traveling Salesman Problem [13], customers belonging to the same *cluster* must be visited contiguously. An extension of these two problems has been introduced in [7], in which a given vehicle can alternate visits between the customers of different clusters, i.e. it is not mandatory to consecutively visit customers of the same cluster. Recently, a new variant of the OP called the Set Orienteering Problem (SOP) was proposed in [2]. In this problem, a single vehicle can visit one customer per cluster at most. To the best of our knowledge, this study is the first to address the CluTOP as defined in our work.

The CluTOP can be used to model many applications in logistic systems and transportation. Some interesting applications were introduced in [1] for the COP, which are still relevant for the CluTOP. An example of these applications is related to the distribution of mass products. In this application, each supply chain contains a set of retailers (customers), and if a carrier agrees to supply a chain with a product, it has to serve all the retailers belonging to this chain. In case where retailers are located in the same area/city, it would be more interesting to represent all these retailers using one vertex node in the graph model (since there is no capacity constraint), which reduces the size of the original problem. As a result, if the retailers belong to different supply chains, the associated vertex node (customer) should belong to different clusters.

The main contributions of our work can be summarized as follows:

- We propose an exact algorithm based on a cutting planes approach. This algorithm includes the implementation of a pre-processing procedure together with the consideration of valid inequalities. These components are introduced in order to reduce the computational-burden of the exact method. Interestingly, these components are deduced by computing cliques on particular graphs that represent incompatibility between customers or clusters.

- We extend our heuristic initially proposed for the COP in [21]. This heuristic explorers both direct representation and giant tours search spaces. Indeed, an Adaptive Large Neighborhood Search (ALNS) generates *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 Beam Search algorithm that incorporates a knapsack-based procedure used to select in the most promising nodes to be expanded in the beam search tree.

- In terms of computational experiments, tests were conducted on benchmarks proposed by [1] to show the efficiency of our methods. Eighty additional instances were solved to optimality by the cutting plane, whereas, the hybrid heuristic succeeded in improving the best solution for 38 instances. Furthermore, we present the results of extensive computational experiments on new proposed CluTOP instances where $m \geq 2$. We report the derivation of very near-optimal solutions for CluTOP instances.

The remainder of this paper is as follows. In Section 2, we describe the new problem and we introduce a mathematical formulation together with valid inequalities. In Section 3, we present our exact approach based on a cutting plane scheme and describe the pre-processing procedure. Section 4 details our hybrid-heuristic. Computational results are presented in Section 5. Finally, concluding remarks are given in Section 6.

2. Mathematical formulation and valid cuts

2.1. Problem description and mathematical formulations

An instance of the CluTOP, I_{CluTOP}, is modeled using a complete undirected graph $G = (V, E)$. The set of vertices is $V = \{1, \ldots, n\} \cup \{0\}$, i.e. a vertex i, where $i > 0$, is associated with each customer in addition to the depot (vertex 0). For each edge $e = (i, j) \in E$ is defined a travel cost denoted, as appropriate, c_e or $c_{(i,j)}$. These costs are assumed to satisfy the triangle inequality and to be symmetric. A set of K clusters $S = \{S_1, S_2, \ldots, S_K\}$ form a cover of $V \backslash \{0\}$ where $\cup_{k=1}^{K} S_k = V \backslash \{0\}$. We note that it is possible to have shared customers between several clusters. A profit P_k is associated with each cluster S_k and collected only if all the customers of the cluster S_k are served. A set of m vehicles is available to serve customers during a limited travel time T_{max}. It is worth mentioning that any vehicle can visit customers from several

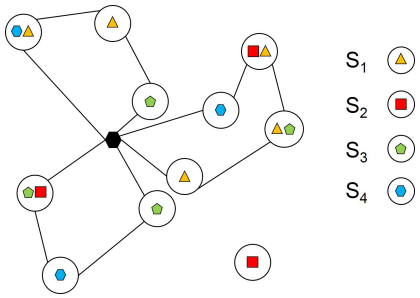

Figure 1: Example of a CluTOP solution

clusters, and the customers of a given cluster can be visited by several vehicles. Furthermore, there is no requirement as to the order of the visits, i.e. a vehicle can alternate visits between customers belonging to different clusters.

Figure 1 shows an example of a feasible solution for a CluTOP instance with 11 customers. In this example, there are four clusters S_1, S_2, S_3 and S_4 which are represented in the figure by a triangle, rectangle, pentagon and hexagon, respectively. Customers are represented by nodes. If a customer belongs to a cluster, the frame that represents this cluster is added inside the node. In this solution, only the customers of clusters S_1, S_3 and S_4 are served.

Before proceeding further, let first consider the following notation. Given U a subset of V, we denote the set of edges with one endpoint in U and one endpoint in $V \backslash U$ by $\delta(U)$. $E(U)$ is used to denote the edges with both endpoints in U. For the ease of notation, when $U = \{i\}$ we will write $\delta(i)$ instead of $\delta(\{i\})$. Finally, $\zeta(i)$ represents the set of clusters to which customer i belongs.

We present a mathematical formulation for the new problem. For that purpose, we introduce the following decision variables :

- z_k : equal to 1 if all customers in cluster $S_k \in S$ are served, 0 otherwise.
- y_{ir} : equal to 1 if vertex $i \in V$ is served by vehicle $r \in \{1, \ldots, m\}$, 0 otherwise.
- x_{er} : equal to 1 if edge $e \in E$ is traversed by vehicle $r \in \{1, \ldots, m\}$, 0 otherwise.

The mathematical model, hereafter denoted by (ILP0), is written as follows:

$$max \sum_{S_k \in S} P_k z_k \tag{1}$$

$$\sum_{r=1}^{m} y_{ir} \leq 1 \quad \forall i \in V \backslash \{0\} \tag{2}$$

$$\sum_{e \in \delta(i)} x_{er} = 2y_{ir} \quad \forall i \in V, r \in \{1, \ldots, m\} \tag{3}$$

$$\sum_{e \in E} c_e x_{er} \leq T_{max} \quad \forall r \in \{1, \ldots, m\} \tag{4}$$

$$\sum_{e \in E(U)} \sum_{r=1}^{m} x_{er} \leq \sum_{i \in U \setminus \{t\}} \sum_{r=1}^{m} y_{ir} \quad \forall U \subseteq V \setminus \{0\}, \quad \forall t \in U \tag{5}$$

$$z_k \leq \sum_{r=1}^{m} y_{ir} \quad \forall S_k \in S, \quad \forall i \in S_k \tag{6}$$

$$z_k \in \{0, 1\} \quad \forall S_k \in S \tag{7}$$

$$x_{er} \in \{0, 1\} \quad \forall e \in E, \quad \forall r \in \{1, \ldots, m\} \tag{8}$$

$$y_{ir} \in \{0, 1\} \quad \forall i \in V, \quad \forall r \in \{1, \ldots, m\} \tag{9}$$

The objective function (1) maximizes the total collected profit. Constraints (2) ensure that a customer is visited by one vehicle at most. It should be specified that this constraint is not defined for the depot. Therefore, the depot can be visited by the m vehicles. Constraints (3) are the flow conservation constraints. Constraints (4) guarantee that the travel time of each vehicle does not exceed T_{max}. Constraints (5) are subtour elimination constraints (SECs). Constraints (6) together with the objective function state that the profit of a cluster is gained only if all of its customers are served. Constraints(7)-(9) define the integrality constraints.

2.2. Valid inequalities

The aim of valid inequalities is to reduce the search space in order to enhance the performance of the resolution of MIP. In this section, we propose valid inequalities based on symmetry breaking and bounding approaches. We also introduce valid inequalities that take advantage of the properties of the CluTOP instance such as inaccessible customers and incompatibility between customers or clusters.

To begin with, we introduce a valid inequality, that states that a customer should not be visited if no cluster among those where it belongs is selected.

$$\sum_{r=1}^{m} y_{ir} \leq \sum_{S_k \in \zeta(i)} z_k \quad \forall i \in V \setminus \{0\} \tag{10}$$

2.2.1. Cuts based on incompatibilities

In this section we compute incompatibilities between components of a $CluTOP$ instance such as customers and clusters. We use undirected graphs to represent the computed incompatibilities and we derive valid inequalities for the $ILP0$. Before proceeding further we recall that:

- A clique in an undirected graph is a subset of vertices that are pairwise adjacent;
- A clique is maximal if it cannot be extended to a bigger one by adding more vertices.

In the following, we propose a set of valid inequalities based on computing subsets of mutually incompatible vertices.

Definition 2.1. Two customers i and j are said to be incompatible if and only if they cannot be visited by the same vehicle due to the travel time constraint, i.e. $c_{(0,i)} + c_{(i,j)} + c_{(j,0)} > T_{max}$. We note that $c_{(i,j)} = c_{(j,i)} \quad \forall (i,j) \in E$ since travel times are symmetric.

In order to represent the different incompatibilities that can exist between each pair of customers, we define the customer-incompatibility graph $G^{inc} = (V, E^{inc})$. The set of arcs E^{inc} is constructed as follows. Let $i, j \in V \backslash \{0\}$, $(i,j) \in E^{inc}$ if and only if i and j are incompatible. Clearly, a clique extracted from the graph G^{inc} includes a set of customers that cannot be served by the same vehicle. Hence, the following proposition holds:

Proposition 2.1. *Let $\mathcal{C}$ denote the set of maximal cliques of the incompatibility graph G^{inc}. The following inequalities:*

$$\sum_{i \in C} y_{ir} \leq 1 \quad \forall C \in \mathcal{C}, \forall r \in \{1, \ldots, m\} \tag{11}$$

are valid for $ILP0$

In the following, we extend the concept of incompatibility between customers to cover the case of clusters.

Definition 2.2. Two clusters S_k and S_l are said incompatible if and only if it is impossible to serve all their customers by using the m available vehicles.

In order to compute the cliques of incompatible clusters, we define the graph of incompatibility $G^{inc}_{cl} = (S, E^{inc}_{cl})$ where E^{inc}_{cl} is constructed as follows. Let $S_k, S_l \in S$, $(S_k, S_l) \in E^{inc}_{cl}$ if and only if S_k and S_l are incompatible. Clearly, a clique extracted from G^{inc}_{cl} includes a set of clusters that cannot be served by using the m available vehicles. Hence the following proposition holds:

Proposition 2.2. *Let $\mathcal{C}_{cl}$ be the set of maximal cliques of the incompatibility graph G^{inc}_{cl}. The following inequalities are valid for $ILP0$.*

$$\sum_{S_k \in C} z_k \leq 1 \quad C \in \mathcal{C}_{cl} \tag{12}$$

2.2.2. Symmetry breaking cuts

In order to reduce the search space, we propose to consider symmetry breaking constraints that eliminate many equivalent solutions. In our model, equivalent solutions occur for example by interchanging any pair of routes. To avoid such configurations, we consider constraints that impose a lexicographical order within the routes. In this paper, we introduce a specific criterion for the CluTOP. We propose associating a score p_i with each customer i. This score is calculated as $p_i = \sum_{k=1}^{K} \rho_{ik}$ where ρ_{ik} is the contribution of customer i to cluster S_k, and it is calculated as $\rho_{ik} = \frac{P_k}{|S_k|}$ if $i \in S_k$ and $\rho_{ik} = 0$ otherwise. The symmetry breaking cut is:

$$\sum_{i \in V \setminus \{0\}} y_{i(r+1)} p_i - \sum_{i \in V \setminus \{0\}} y_{ir} p_i \leq 0 \quad r = \{1, \ldots, m-1\} \tag{13}$$

2.2.3. Bounding cuts

We propose two bounding valid inequalities. The first is based on computing an upper bound on the total profit while the second is based on computing a lower bound on the number of selected clusters.

Definition 2.3. Two clusters S_k and S_l are said to be compatible if and only if a feasible solution exists using the m available vehicles, where all their customers are served.

Let Φ be a collection of subsets of mutually compatible clusters and $\phi^* \in \Phi$ such that:

$$\sum_{l:S_l \in \phi^*} P_l = \max_{\phi \in \Phi} \sum_{k:S_k \in \phi} P_k$$

Clearly, $\sum_{l:S_l \in \phi^*} P_l$ is a valid upper bound on the total profit that might be collected. Therefore, the following valid inequality holds:

$$\sum_{k=1}^{K} P_k z_k \leq \sum_{l:S_l \in \phi^*} P_l \tag{14}$$

Interestingly, computing ϕ^* turns out to be the maximum weighted independent set in the graph G_{cl}^{inc}, where the weight of vertex S_k is set to P_k.

The second bounding valid inequality is described as follows. Let LB_{cl} be a lower bound value on the number of clusters that should be served on a feasible solution of I_{CluTOP} with at least Pr as total profit. Therefore, the following constraint holds:

$$\sum_{S_k \in S} z_k \geq LB_{cl} \tag{15}$$

3. Cutting plane algorithm

In this section, we describe our exact algorithm. This algorithm is based on cutting plane approach that includes a preprocessing procedure and a heuristic approach that aims to repair unfeasible solutions.

3.1. Global scheme

We start by describing the global scheme of our algorithm. In order to solve CluTOP, we are interested in solving $ILP0$. Moreover, we include the valid inequalities (10)-(13). Our algorithm is based on the cutting plane approach. Indeed, a MIP-solver is used to solve to optimality a relaxation version of $ILP0$ (called $ILP1$) where subtour elimination constraints are relaxed. Clearly, an integer solution (S^*_{ILP1}) is obtained . If this solution does not contain any subtour, then the solution is optimal for $ILP0$, otherwise the set of subtours are extracted and necessary subtour elimination constraints are generated and added to $ILP1$. This process is iteratively applied until either an optimal solution without subtours is found or the time limit has been reached.

In order to verify if any subtours exist, we use a Depth-First Search algorithm (DFS) to detect connected components in an undirected graph. The DFS should detect two types of connected components. The first type is the components that contain the depot, hereafter refered to as main tours. The other connected components are considered as subtours (tours separated from the depot). Once the subtours are extracted, suitable subtour elimination constraints are generated and added to the model. In our work, we use more general SECs called Generalized Subtour Elimination Constraints (GSECs) proposed in [11]. The GSECs are defined as follows:

$$\sum_{e\in\delta(U)} x_{er} \geq 2y_{ir}, \quad \forall U \subset V, 0 \in U, \forall i \in V\backslash U, r \in \{1,\ldots,m\} \tag{16}$$

$$\sum_{e\in E(U)} x_{er} \leq \sum_{i\in U} y_{ir} - y_{jr}, \quad \forall U \subset V, 0 \in U, \forall j \in V\backslash U, r \in \{1,\ldots,m\} \tag{17}$$

$$\sum_{e\in E(U)} x_{er} \leq \sum_{i\in U} y_{ir} - y_{jr}, \quad \forall U \subseteq V\backslash\{0\}, \forall j \in U, r \in \{1,\ldots,m\} \tag{18}$$

Moreover for a better performance the cutting plane algorithm includes the following features:

- A pre-processing procedure that aims to reduce the number of decision variables and to initialize and fill the customer and cluster incompatibility graphs with more edges.
- A local repair solution heuristic that aims to repair S^*_{ILP1} if it contains any subtour.

Furthermore, the adopted branching rules for the resolution of $ILP1$, prioritize z_i first, then y_{ir} and finally x_{er}. This can be motivated by the fact that the objective function in CluTOP aims to maximize the collected profit from the visited clusters [8].

The global scheme of our model is described in Algorithm 1.

Algorithm 1: GLOBAL SCHEME

Input: instance I
Output: solution for I

```
Construct the model ILP0
Calculate a feasible solution for I (see Section 4)
Execute the pre-processing procedure(see Section 3.3)
Add the computed valid inequalities and relax constraints (5) to obtain
  the model ILP1
repeat
    Compute S*_ILP1 the optimal solution of ILP1
    if S*_ILP1 does not contain any subtour then
        Set S*_ILP1 as optimal for CluTOP
    else
        Calculate all subtours and add the corresponding GSECs to
          ILP1
        From S*_ILP1 construct a partial solution using only the main
          tours
        Repair the partial solution (See Section 3.2)
until (S*_ILP1 is optimal for CluTOP or time expired)
```

3.2. Solution repair

Recall that if S^*_{ILP1} is not optimal for $ILP0$, then it is composed of two types of tours: the main tours and the subtours which are not related to the depot. A trivial feasible solution X can be constructed from S^*_{ILP1} using only main tours. Unfortunately, this partial solution is often of poor quality. This is mainly due to the nature of CluTOP in which the profit of a given cluster is collected only if all of its customers are served. As a result, if at least one customer of a given cluster belongs to one of the subtours, the whole profit of the cluster will be discarded. In this section, we propose a greedy procedure to repair the solution X extracted from S^*_{ILP1}. First, the clusters of S^*_{ILP1} are sorted according to a non-increasing order of the following criterion: $\frac{P_k}{N(S_k)+1}$, where $N(S_k)$ is the number of customers of cluster S_k located in subtours. This criterion favors, on the one hand, clusters with higher profits, and on the other hand, those with a small number of customers located in subtours. The insertion of customers in the current solution X is carried out iteratively cluster by cluster. The customers of a given cluster are inserted one by one using a best insertion approach. If the procedure fails to insert at least one customer

of a given cluster, then all its customers will be omitted from the solution X except those shared with already inserted clusters.

3.3. Pre-processing phase

In this section, we describe the pre-processing procedure that aims to fix some decision variables and to compute incompatibility graphs.

Below, we denote by:

- $UB(I)$: the value of the upper bound delivered by the cutting plane procedure within a small time budget for a CluTOP instance I,
- $LB(I)$: the value delivered by the hybrid heuristic described in section 4.

3.3.1. Inaccessible components

Definition 3.1. A customer i is considered to be inaccessible if the tour that starts and ends at the depot and exclusively serves this customer has a length greater than T_{max}, i.e. $c_{(0,i)} + c_{(i,0)} > T_{max}$.

Definition 3.2. A cluster is said to be inaccessible if it is impossible to visit all of its customers using all available vehicles.

On the basis of these definitions, we should fix some of the decision variables y_{ir} and z_k as proposed in the following two equations. The first is related to the inaccessible customers, while the second concerns the inaccessible clusters.

$$y_{ir} = 0 \quad \forall i \in V \backslash \{0\}, \text{ and } c_{(0,i)} + c_{(i,0)} > T_{max}, \forall r = 1 \dots m \tag{19}$$

$$z_k = 0 \quad \forall k = 1, \dots, K \text{ and } S_k \text{ is inaccessible} \tag{20}$$

At this point it should be specified that checking whether a customer is inaccessible or not requires $O(1)$-time. However, in the case of a cluster, this procedure requires solving a multiple travel salesman problem (MTSP). This problem is NP-complete even in the case where only one salesman is available. For this reason, we consider a trivial relaxation instead of MTSP. This relaxation is based on exploring inaccessible customers. Indeed, if a customer is inaccessible then all clusters that share it could be considered as inaccessible.

3.3.2. Mandatory clusters

The basic idea to compute mandatory clusters is as follows. Given an instance I of CluTOP, an instance $\bar{I}_k$ is derived from I by ignoring the cluster S_k. The cluster S_k is considered as mandatory if $UB(I_k) < LB(I)$. Therefore, the variable z_k should be fixed to 1.

3.3.3. Useful pre-computations

In addition to these pre-processing features, we perform some pre-computations such as the incompatibility graphs and some specific lower and upper bounds that are useful to generate the considered cuts.

In the following we describe the computation of the incompatibility-graphs. The customers-incompatibility graph is calculated as follows: for each vertex in the graph, we calculate the maximal clique containing this vertex using meta-heuristic proposed in [6].

However, the construction of clusters-incompatibility graph requires more computational effort. Recall that computing incompatibility between a pair of clusters needs to solve mTSP problem. Solving such problem several times can be very time consuming. We therefore propose to proceed heuristically to deduce incompatibilities. To do this, let us introduce the following proposition.

Proposition 3.1. *Let G^{inc}_{kl} be the subgraph induced in G^{inc} by the subset $S_k \cup S_l$. Let C_{kl} be a maximum clique extracted from G^{inc}_{lk}. If $|C_{kl}| > m$, then S_k and S_l are incompatible.*

To solve the Maximum Clique Problem (MCP), we propose to use the exact method proposed in [15]. Although this problem is NP-hard, algorithms used are quite fast when dealing with small instances (up to 60 vertices in our case).

To further enhance the density of the incompatibility graph between clusters, we propose the following improvement. Given an instance I, two clusters S_k and S_l, the aim is to solve the sub-instance in which we consider only the customers of these two clusters. We denote this sub-instance by I_{kl}. Therefore the following proposition holds:

Proposition 3.2. *If $UB(I_{kl}) < P_k + P_l$ then S_k and S_l are incompatible.*

This second phase is only applied on couple of clusters that have not been yet proved to be incompatible during the first phase. During experiments, it was showed that solving CluTOP on a couple of clusters is quite fast compared to solving the whole instance.

Algorithm 2 details the pre-processing phase.

4. Heuristic Scheme

In this section, we describe a generalization of our heuristic presented in [21] initially designed for the COP. It is based on the *order first-cluster second* approach [18] and consists of two phases: the first one is the ordering phase in which a *giant tour* covering all the customers is constructed. The second phase is a splitting procedure that extracts the optimal solution from a given giant tour while respecting the predefined sequencing of its customers.

4.1. Split procedure principle

Given a giant tour $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$. that covers all the customers, the splitting procedure aims at find a subset of clusters that maximizes the total

Algorithm 2: PREPROCESSING

Input: instance I, Lower bound LB

Output: C The set of valid inequalities; the set of inaccessible customers and clusters; the set of mandatory clusters

Initialize C by the symmetry breaking and bounding valid inequalities(See Sections 2.2.2 and 2.2.3)

Calculate inaccessible customers and clusters (See Section 3.3.1)

Calculate mandatory clusters (See Section 3.3.2)

Calculate $G^{inc}(V, E^{inc})$ the incompatibility graph of customers

Calculate maximal cliques in $G^{inc}(V, E^{inc})$ and derive the valid inequalities based on incompatibilities between customers Add these valid inequalities to the set C

Initialize $G^{inc}_{cl}(S, E^{inc}_{cl})$ the graph of incompatibility between clusters

foreach $((S_k, S_l) \in S^2)$ **do**

    Extract the graph G^{inc}_{kl} the sub graph induced in G^{inc} by the subset $S_k \cup S_l$

    $\mathcal{Q}_{kl} \leftarrow$ calculate maximum clique in G^{inc}_{kl}

    if $(|\mathcal{Q}_{kl}| > m)$ **then** Add (S_k, S_l) to E^{inc}_{cl}

    else if $(UB(I_{kl}) < P_k + P_l)$ **then**

        Add (S_k, S_l) to E^{inc}_{cl}

Calculate maximal cliques in G^{inc}_{kl} and derive the valid inequalities based on incompatibilities between clusters

Add these valid inequalities to the set C

collected profit with respect to the order of the giant tour and the time limit. The procedure relies on a branch-and-bound scheme in which, we embedded a knapsack-based upper bound to fathom inferior nodes and a feasibility test to discard unfeasible nodes. The branching scheme consists in enumerating all of the subsets of clusters. Starting from the root node with an arbitrarily ordered list of potential clusters, a descendant node is derived either by selecting or discarding the first cluster in the list. By applying this process on all the potential clusters, this leads to a binary search tree with at most $2^{K+1} - 1$ nodes. In the following, for each node η in the search tree, we use the sets S^{η}_{p}, S^{η}_{s} and S^{η}_{r} to denote the potential clusters, the selected clusters and removed clusters, respectively. Exhaustive enumeration of all subsets of clusters may be both costly and inefficient. To minimize this drawback, we propose to use a beam search technique. This technique helps to keep computational times under a known value. The main idea is to explore the search tree in Breath First Search (BFS), and to pickup a limited number of nodes to expand at each level. Unfortunately, this new scheme does not guarantee that the solution found

is optimal. Hence, we propose to use the upper bound described in Section 4.1.3 as a selection criterion so that only promising nodes will be selected at each level of the tree. Another important aspect is the number of nodes selected at each level. This parameter was fixed after preliminary experimentation at K nodes per level.

4.1.1. Feasibility check

A feasibility check (FC) is performed every time a potential cluster is added to the set of selected clusters S_s^η. A given node is feasible if all the customers of its selected clusters can be visited using m vehicles at most. To do this, we consider the partial sequence $\pi^\eta = (\pi_1, \ldots, \pi_{|\pi^\eta|})$ extracted from the giant tour π by keeping only customers of the selected clusters.

The procedure FC can be described as follows. At each iteration i, the i^{th} customer in the partial sequence π^η, is inserted at the end of the last route. If this insertion fails, either because the travel time exceeds T_{max} or no route has yet been initialized, the customer is inserted in a new initialized route. This process is reiterated until all customers of the sequence π^η are served. Thanks to the triangular inequality, the number of visited customers per route is maximized and the number of used vehicles (m_σ) is then minimized. Therefore, if $m_{\pi^\eta} > m$, then the partial solution is unfeasible and the node should be pruned. The complexity of this procedure is $O(n)$.

4.1.2. TOP-based Relaxation

In [21], we presented a relaxation scheme for the COP. The generalization for the case of the $CluTOP$ is trivial. The definition as well as the proposition and its proof are all given so that the paper will be self contained.

Definition 4.1. Given a $CluTOP$ instance I with its undirected graph G=(V,E), we define a TOP instance I_{TOP} defined by the same graph G=(V,E). The pseudo profit p_j of each customer in $j \in I_{TOP}$ is calculated as showed in Section 2.2.2. The maximal travel time of I_{TOP} is the same as instance I, which is T_{max}. We also define the following notation:

- $P^*(I)$ is used to denote the optimal objective value of instance I.
- Let $S' \subseteq S$ be a subset of clusters. We denote the sum of their profits by $P_{CluTOP}(S')$. Hence, $P_{CluTOP}(S') = \sum_{k:S_k \in S'} P_k$.
- Let $V' \in V$ be a subset of customers in I_{TOP}. We denote the sum of their profits by $P_{TOP}(V')$. Hence, $P_{TOP}(V') = \sum_{j \in V'} p_j$.

Proposition 4.1. *The optimal objective value of the associated instance I_{TOP} represents an upper bound on the profit of I ($P^*(I_{TOP}) \geq P^*(I)$).*

Proof. Let S^* be the set of clusters of the optimal solution of $CluTOP$ instance I. The total collected profit is calculated as $P_{CluTOP}(S^*) = \sum_{i:S_k \in S^*} P_k = P^*(I)$. On the other hand, let V^* be the set of customers of S^*. It is obvious that the optimal solution of I_{CluTOP} is feasible for I_{TOP} and its profit is $P_{TOP}(V^*) =$

$\sum_{j \in V^*} p_j$. We also denote the optimal objective value for I_{TOP} by $P^*(I_{TOP})$. We have,

$$\begin{aligned} P_{TOP}(V^*) = \sum_{j \in V^*} p_j &= \sum_{j \in V^*} \sum_{k: j \in S_k} \frac{P_k}{|S_k|} \\ &= \sum_{j \in V^*} \sum_{k: j \in S_k \text{ and } S_k \in S^*} \frac{P_k}{|S_k|} + \sum_{j \in V^*} \sum_{k: j \in S_k \text{ and } S_k \notin S^*} \frac{P_i}{|S_i|} \\ &= P_{CluTOP}(S^*) + \sum_{j \in V^*} \sum_{k: j \in S_k \text{ and } S_k \notin S^*} \frac{P_k}{|S_k|} \\ &= P^*(I) + \sum_{j \in V^*} \sum_{k: j \in S_k \text{ and } S_i \notin S^*} \frac{P_k}{|S_k|} \end{aligned} \quad (21)$$

As a result, the optimal solution for I is feasible for the I_{TOP}. Furthermore, $P^*(I) \leq P_{TOP}(V^*) \leq P^*(I_{TOP})$. □

4.1.3. Knapsack-based upper bound

We propose in this paper an upper bound based on the Fractional Knapsack Problem. We first extend Proposition 4.1 to cover the case of giant tours $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$. Since a giant tour π imposes an order of visits among the customers of I, it can be seen as a derived instance I' in which only edges that respects this order are considered. The following corollary holds.

Corollary 4.1. *The optimal objective value of the associated instance I_{TOP} w.r.t a given giant tour π represents an upper bound on the profit of I w.r.t π.*

Given now a node η in the search tree, we assume that the partial solution retrieved by the procedure FC is feasible. Otherwise the node should be pruned. We consider the following Knapsack instance I_{FKSP} in which we associate an item with each potential customer. A customer is considered as potential if it belongs to at least one of the potential clusters S_p^η and does not belong to any of the selected clusters S_s^η.

The profit of an item π_j is calculated using Definition 4.1. Note that to calculate these profits in a node η, we consider only contributions related to potential clusters S_p^η and we discard contributions related to removed clusters S_r^η. As a result, the profit of π_j is calculated as follows: $p_{\pi_j}^\eta = \sum_{i: \pi_j \in S_i \text{ and } S_i \in S_p^\eta} \frac{P_i}{|S_i^\eta|}$, where $|S_i^\eta|$ is the number of potential customers belonging to cluster S_i in node η.

The weight $w_{\pi_j}^\eta$ of the item π_j is the minimal insertion cost. Let I_j^η be the set of all the insertion positions of π_j, where each position is defined by one predecessor and one 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 size of the knapsack W^η, we proceed as follows. We consider the sub-sequence formed by the customers of the selected clusters S_s^η. Knowing that the node is feasible, i.e. all the customers in the sub-sequence can be

visited using at most m vehicles, the aim is to find the set of tours that minimizes the total distance. This can be seen as solving a Distance Constrained VRP with a limited fleet on a given permutation of customers. This problem can be efficiently solved by applying a modified version of the split procedure proposed in [4] for VRP. In our study we use an effcient implementation proposed by Vidal in [20] with $\mathcal{O}(nm)$ time and space complexity. Assuming now that C^η is the total distance, W^η is simply modeled as the residual distance, i.e. $W^\eta = mT_{max} - C^\eta$.

Proposition 4.2. *Given a giant tour π and a node η in the search tree, the optimal objective value of the I_{FKSP} is an upper bound on the optimal objective value of the I.*

Proof. Given a giant tour π covering all the customers of I and a node η. We construct a knapsack instance I_{FKSP} in which, each item π_j has a weight w_j^η and a profit $p_{\pi_j}^\eta$.

Assume π^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 π^η. According to the definition of the minimal cost insertion, it is obvious 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_{TOP} while considering π and η. According to Corollary 4.1, I_{FKSP} is also an upper bound on I_{CluTOP} while considering π and η. □

4.1.4. Local search procedure

We propose to improve the splitting procedure by integrating a Local Search heuristic (LS). The LS uses some relevant information in the nodes of the beam search in order to efficiently explore the search space. When LS is called in a node η, it is applied only on the selected and the potential set of clusters $S_s^\eta \cup S_p^\eta$. In this way, the LS focuses on a reduced part of the search space, and hopefully, it succeeds easily and quickly finding a new global best solution. An initial solution is constructed from the customers of the selected clusters using the splitting procedure for the Distance Constrained VRP with a Limited Fleet [20] (see Section 4.1.3). After that, a potential cluster is randomly selected and its unrouted customers are inserted in the solution using a best insertion approach without considering the time limit constraint. The Lin-Kernighan heuristic [14] is then applied on each tour separately. If it fails to find a TSP solution with a travel time equal or less than T_{max} for at least one tour, the insertion of the potential clusters is considered as unfeasible. This process is repeated on all of the potential clusters.

Algorithm 3 provides a pseudo code of the split procedure. We use in Algorithm 3 two priority queues where the Knapsack upper bound is used as a priority criterion (See Section 4.1.3) (line 1), the first is called *currList* that contains the nodes of the current level, while the second list *tmpList* contains the nodes to be explored in the next level of the tree. The lower bound LB is initialized by the current best solution of the global heuristic.

Algorithm 3: SPLIT

Input: giant tour GT, Lower bound LB
Output: best solution X_{best}
Data: Priority queue of size K: $currList$, $tmpList$
Nodes: e, e_1, e_2

Initialization: $Order \leftarrow$ array of K clusters arbitrarily ordered
$L \leftarrow 1$ (current level)
$e \leftarrow rootNode$
$currList.enqueue(e)$
while $(currList \neq \emptyset \text{ and } L \leq K)$ **do**
  $e \leftarrow currList.dequeue()$
  Expand e into two nodes e_1 and e_2 by branching on cluster $Order[L]$ (Section 4.1)
  foreach $(e \in \{e_1, e_2\})$ **do**
    if (e is infeasible) **then continue** (Section 4.1.1)
    if (Knapsack UB of $(e) \leq LB$) **then continue** (Section 4.1.3)
    $tmpList.enqueue(e)$
    Extract solution X from e
    Apply LS on X (Section 4.1.4)
    if $(Eval(X) > Eval(X_{best}))$ **then**
      $X_{best} \leftarrow X$
      if $(Eval(X) > LB)$ **then** $LB \leftarrow Eval(X)$
  if $(currList = \emptyset)$ **then**
    $currList \leftarrow tmpList$
    $tmpList \leftarrow \emptyset$
    Increment L
Select the best node e in $currList$ and Extract solution X_{best}
return X_{best}

4.2. Global Algorithm

The ALNS scheme was first proposed by [17] for routing and scheduling problems. Since then, the ALNS has been widely used to solve different variants of VRPs and has proven to be an efficient framework. The main characteristic of this metaheuristic is the use of several insertion and removal operators during the search process. An operator is a heuristic capable of exploring a large neighborhood in a short time. At each iteration, a pair of insertion and removal operators are randomly chosen. In this way, the ALNS is able to explore several neighborhoods during the search process. At the same time, statistics are gathered in order to estimate the contribution of each operator to the solution progress. These statistics are then used to promote the most effective operators. This characteristic offers the ALNS the flexibility to tackle a wide range of instances.

Our ALNS scheme includes one destruction operator that selects a random

number of clusters and removes their customers from the solution. Note that customers shared with other clusters remain in the solution. The number of clusters to remove d_{max} (line 6) is adjusted during the search. It is incremented after each iteration without improvement (line 12) and reset to the initial value once a new best solution is found (line 12). After experimental tests, the initial value is set to 3.

Regarding the solution repair (line 8-9), unrouted clusters are iteratively inserted into the current solution one by one until no cluster is left or no further insertions are possible. Once a cluster is randomly selected, its unrouted customers are identified (some of its customers shared with other clusters would already be in the solution) and inserted using one of the following insertion operators.

1. Best Insertion Operator. All insertion positions are evaluated and the best move is selected.
2. Random Best Insertion Operator. A customer is randomly selected and inserted in the position with the minimum cost.
3. 2-Regret Insertion Operator. For each customer, the two insertion positions with the minimum cost are identified and the gap is calculated. The customer with the maximum gap is inserted in its best position.

We use a local search operator called 2-opt to improve the travel time of the current solution (line 7). This operator is called at each iteration between the removal and the insertion operator.

In our ALNS, each insertion operator is associated with a value called *weight*. The selection of an insertion operator depends on its weight: the larger is the weight, the more probable that it will be selected. These weights, initialized by the same value, are dynamically adjusted during the search process according to the performance of each operator (line 14). The goal is to assign larger weights to the most effective operators. The update of each weight depends on the quality of the obtained solution. If the selected operator yields a new best solution, it is assigned a larger score, whereas it is assigned a low score if the new solution is worse than the current one. A medium score is assigned if the new solution is better than the current one but still not better than the best solution. For more details about the update procedure, the reader is referred to Pisinger and Ropke [17]. Algorithm 4 describes the global scheme of our heuristic.

The number of iterations of the ALNS is fixed at n (line 5-14). The best solution found by the ALNS is used to construct a giant tour (line 15). The construction is done by concatenating the tours and then inserting the customers of the unrouted clusters in random positions of the giant tour. The latter is provided to the splitting procedure in the purpose of improving the quality of the best solution (line 16). This process (ALNS + SPLIT) is iterated until one of the two stop conditions is verified: either $log(n * K)$ iterations are performed without improvement, or the total number of iterations exceeds the value n.

Algorithm 4: GLOBAL SCHEME

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

$X_{best} \leftarrow X$
repeat
  Initialize d_{max}
  $X' \leftarrow X$
  for $(i = 1; i \leq n; i++)$ **do**
    Remove d_{max} clusters from X'
    Apply 2-opt on X
    Select an insertion operator i
    Apply i on X'
    if $(Eval(X') > Eval(X))$ **then**
      $X \leftarrow X'$
      Increment d_{max}
    else Reset d_{max}
    Update weights using the adaptive weight adjustment procedure
  Construct a giant tour GT from X
  $X \leftarrow SPLIT(GT, Eval(X_{best}))$(See Algorithm 3)
  if $(Eval(X) > Eval(X_{best}))$ **then** $X_{best} \leftarrow X$
until *(stop condition is reached)*
return X_{best}

5. Computational tests

In this section, we present a detailed description of the tests we made in order to evaluate the performance of our algorithms. Our algorithms are coded in C++ using the Standard Template Library (STL) for data structures. Experiments were conducted on a Linux OS 64-bit computer with Intel Xeon(R) E2-2670 16-core CPU@2.60 GHz and 128 gigabytes RAM. The cutting plane is implemented using Cplex 12.6 and Concert technology.

We tested our algorithms on two different problem sets. The first set (Set A) concerns the instance introduced in [1] for the case of a single vehicle. We note that methods proposed in [1] were tested on a 64-bit computer with Intel Xeon W3680 six-core CPU@3.33 GHz. The second set (Set B) is related to the multiple vehicles instances. In the following, we provide a detailed description of these sets and we report the results of our computational experiments.

5.1. Set A: single vehicle problem set

In this section, we focus on the single vehicle case of the *CluTOP*. We propose a general comparison between our methods and the methods from the literature presented in [1].

To do this, we use benchmark instances introduced in [1].

5.1.1. Description of the instances

The benchmark is derived from 57 instances of TSPLIB with a number of vertices ranging from 42 to 532. For each base instance of TSPLIB, a set of derived instances for the COP is constructed according to different values assigned to the following parameters:

1. Number of clusters: the number of clusters K takes values of 10, 15, 20 or 25. Clusters were generated in order to have approximately the same number of customers. We note that three additional values (50, 75 and 100) were considered for the largest TSPLIB instance (att532).
2. Profits of clusters : the profits of clusters are generated as follows . First, a profit is assigned to each customer and the profit of a given cluster is then calculated as the sum of the profits of its customers. Two patterns are used to generate the profits of the customers [10]. In the first one the profit of each customer is equal to one. In the second pattern, the profit of each customer is generated using the formula $1+(7141j+73)mod(100)$, where j is the index of the customer.
3. T_{max}: Given TSP^* the optimal value of TSP over all vertices of the base instance, the value of T_{max} is set at $\theta * TSP^*$, where θ takes two possible values: $q2 = \frac{1}{2}$ and $q3 = \frac{3}{4}$.

As a result, 16 different instances are derived from each TSP benchmark instance except for *att532* where 24 new instances were derived. As a result, 924 instances are used to evaluate the proposed methods. The instances can be found at the following URL: http://or-brescia.unibs.it/. For a detailed description of instance generation, the reader can refer to Angelelli et al. [1].

5.1.2. Performance of the exact method

We conducted a set of comparisons of our exact algorithm with the two branch-and-cut algorithms namely COP-BASIC and COP-CUT of Angelelli et al. [1]. In Table 1, we present the results of the three exact methods. For each method, we report:

- $\#Opt$: the number of times for which it yields the optimal solution within a time limit of 3600 seconds.
- $OptGap$: the average percentage deviation optimal gap that is computed as $\frac{UB-LB}{UB}$, where UB and LB are the upper bound and the lower bound found by each method respectively.
- CPU : average CPU time in seconds.

From Table 1, we observe that:

- The cutting plane algorithm succeeds in optimally solving more instances than COP-BASIC and COP-CUT, with 643 instances. COP-BASIC solved 602 instances and COP-CUT solved 558. Interestingly, our algorithm reached the optimilaty gap of 0.043, while requiring less computational

time. This gap jumps to 0.09 and 0.12 for COP-BASIC and COP-CUT, respectively.

- The three algorithms succeed to solve all the instances with a number of customers less than 100 except for class $pr76$, where COP-BASIC fails to close one (1) instance, whereas the cutting plane fails to close one (1) instance from the class $kroC100$.

- Regarding larger instances, we can observe that the performance of the exact methods depends on the class of instances. For the classes of instances $ch130$, $ch150$, $kroA150$ and $kroB150$, COP-BASIC and COP-CUT struggle to find the optimal solution, whereas our cutting plane succeeds to close a large number of them. On the other hand, the performance of our algorithm decreases for the classes $pr226$, $pr264$ compared to COP-BASIC and COP-CUT.

Table 1: Performance of the cutting plane

Class	COP-BASIC			COP-CUT			Cutting plane		
	$\#OPT$	$OptGap$	CPU	$\#OPT$	$OptGap$	CPU	$\#OPT$	$OptGap$	CPU
$dantzig42$	16	0	8.13	16	0	0.61	16	0	2.90
$swiss42$	16	0	6.58	16	0	0.95	16	0	4.08
$att48$	16	0	13.18	16	0	8.45	16	0	34.80
$gr48$	16	0	11.88	16	0	4.86	16	0	12.46
$hk48$	16	0	10.87	16	0	5.54	16	0	12.42
$eil51$	16	0	16.71	16	0	6.76	16	0	16.56
$berlin52$	16	0	12.02	16	0	2.30	16	0	9.96
$brazil58$	16	0	16.44	16	0	5.47	16	0	26.67
$st70$	16	0	52.68	16	0	57.70	16	0	95.71
$eil76$	16	0	20.16	16	0	18.17	16	0	25.83
$pr76$	16	0	94.41	15	0.005	501.65	16	0	196.43
$gr96$	16	0	34.80	16	0	23.26	16	0	37.23
$rat99$	16	0	232.35	16	0	93.09	16	0	159.30
$kroA100$	14	0.037	1112.48	16	0	279.00	16	0	240.96
$kroB100$	11	0.070	1386.00	16	0	468.64	16	0	272.45
$kroC100$	11	0.072	1967.99	16	0	602.79	15	0.005	746.34
$kroD100$	14	0.021	1613.80	16	0	647.23	16	0	581.61
$kroE100$	13	0.028	1107.84	15	0.021	562.51	16	0	179.06
$rd100$	10	0.038	1702.83	16	0	537.00	16	0	225.11
$eil101$	16	0	93.98	16	0	50.94	16	0	115.96
$lin105$	16	0	104.36	16	0	59.40	16	0	72.66
$pr107$	16	0	171.29	16	0	61.62	15	0.019	332.52
$gr120$	10	0.048	1854.87	15	0.018	948.18	15	0.006	688.78
$pr124$	16	0	139.67	16	0	78.51	16	0	100.51
$bier127$	16	0	283.76	16	0	97.43	16	0	92.16
$ch130$	5	0.117	2717.58	12	0.026	1844.03	15	0.0042	767.34
$pr136$	11	0.046	2024.14	16	0	765.13	16	0	326.29
$gr137$	16	0	144.86	16	0	62.89	16	0	149.90
$pr144$	16	0	229.02	16	0	119.39	16	0	135.73
$ch150$	1	0.372	3438.73	0	0.444	3600.71	9	0.0584	2098.01
$kroA150$	2	0.325	3488.52	3	0.328	3232.80	10	0.0583	1713.36
$kroB150$	1	0.333	3591.24	1	0.346	3571.16	11	0.0563	1913.78
$pr152$	16	0	266.37	16	0	129.95	16	0	559.19
$u159$	13	0.013	923.46	16	0	442.11	14	0.0109	1185.79
$si175$	16	0	844.04	16	0	424.51	16	0	315.50
$brg180$	16	0	597.49	16	0	141.47	16	0	155.60
$rat195$	5	0.075	3020.09	7	0.081	3003.96	10	0.0447	1757.66
$d198$	16	0	179.49	14	0.014	1035.47	12	0.0332	1431.59
$kroA200$	0	0.625	3600.76	0	0.746	3600.97	6	0.1473	2562.21

continued on next page

Class	COP-BASIC			COP-CUT			Cutting plane		
	$\#OPT$	$OptGap$	CPU	$\#OPT$	$OptGap$	CPU	$\#OPT$	$OptGap$	CPU
$kroB200$	2	0.456	3471.47	0	0.642	3600.41	6	0.1767	2550.03
$gr202$	13	0.013	814.35	13	0.014	1255.61	16	0	557.13
$ts225$	0	0.369	3600.52	0	0.712	3600.40	0	0.2729	3600.73
$tsp225$	1	0.262	3504.62	3	0.148	3335.69	10	0.0681	2461.35
$pr226$	10	0.110	1782.04	14	0.019	1485.83	5	0.1441	3066.90
$gr229$	16	0	299.62	13	0.021	1645.25	16	0	1063.57
$gil262$	0	0.614	3601.48	0	0.918	3601.64	4	0.1986	3084.40
$pr264$	8	0.066	2499.76	10	0.175	2105.14	2	0.3496	3155.57
$a280$	1	0.176	3415.74	1	0.235	3464.00	2	0.1768	3357.49
$pr299$	1	0.176	3569.61	2	0.548	3554.80	1	0.2837	3449.92
$lin318$	1	0.153	3543.92	0	0.531	3600.58	1	0.1547	3430.73
$Mean$	558	0.092	1344.76	602	0.120	1166.92	643	0.0454	982.64

According to the performance of the three exact methods, we identify three categories of classes of instances. The first category (Category 1) is composed of the classes where our exact method outperforms COP-BASIC and COP-CUT, namely $ch150$, $kroA150$, $kroB150$, $kroA200$, $kroB200$, and $tsp225$. Indeed, our exact method solved 52 instances. On the other hand COP-BASIC and COP-CUT failed to solve 89 out of 96 instances. Interestingly, Angelleli et al. pointed out the poor performance of their methods on classes $kroA200$, $kroB200$. They highlighted the fact that their methods failed to achieve good feasible solutions. At this point, it is noteworthy to indicate that our exact method and COP-CUT yield no null profit-feasible solutions on all the instances of Category 1. However, COP-BASIC failed to retrieve such feasible solutions 16 times. The second category (Category 2) contains the classes $pr226$, $pr264$. For these classes, COP-BASIC or/and COP-CUT outperform our cutting plane. Actually, COP-CUT (respc. COP-BASIC) solved 24 (resp. 18) out of 32 instances to optimality, whereas our exact method solved only 7 instances. This is mainly related to the distribution of customers in these instances, which is in the form of sub-structures like meshes or even a set of co-linear points. Pferschy et al. pointed out in [16] that these type of instances are hard to solve using the cutting plane approach, since they are relatively unstable in terms of computational times, number of iterations, and the number of subtours generated in each iteration as well. Finally, the third category (Category 3) is composed of all the other classes where all the methods of Angelleli et al. [1] on the one hand, and our exact method on the other, present the same either good or poor performance.

In order to provide more detailed picture of the performance of the different exact methods on Category 1 and Category 2, in Table 2, we present a more detailed comparison between them. In this table, we report the average percentage gap of the upper bound (resp. lower bound) delivered by our exact method with respect to those obtained by COP-BASIC and COP-CUT. These values are given in columns $UBGAP$ (resp. $LBGAP$).

Table 2: GAP Cutting plane vs. the literature

Class	UB GAP		LB GAP	
	C-P vs. C-B	C-P vs. C-C	C-P vs. C-B	C-P vs. C-C
*ch*150	−0.447	−0.454	0.257	0.131
*kroA*150	−0.447	−0.458	0.618	0.080
*kroB*150	−0.386	−0.433	0.135	0.092
*kroA*200	−0.504	−0.678	0.622	0.321
*kroB*200	−0.484	−0.503	0.435	0.124
*tsp*225	−0.037	−0.095	0.056	0.181
*pr*226	0.041	−0.018	−0.111	−0.043
*pr*264	0.155	0.174	−0.032	−0.193

On the basis of Table 2, we observe that:

- The poor performance of the exact methods presented in Angelleli et al. [1] is explained not only by the quality of the feasible solution (as suggested by [1]) but also by the weakness of the upper bound. Indeed, our exact method improved the feasible solution delivered by COP-BASIC and COP-CUT by 35.4% and 15.5%, respectively. Surprisingly, the improvement of the quality of the upper bounds is much larger reaching 67.8% compared to COP-CUT upper bound.
- The weak results of our method on the instances in Category 2 is due to the poor performance of both the lower and upper bound compared to Angelleli et al. [1].

5.1.3. Performance of the heuristic method

We propose in the following to verify the performance of the hybrid heuristic. We compared our method with a tabu search based heuristic called COP-TABU, which was proposed in [1]. Three variants of COP-TABU have been implemented: COP-TABU-*Basic*, COP-TABU-*Multistart* and COP-TABU-*Reactive*. Tests were conducted on all the 57 classes of the benchmark [1] (924 instances). We consider a multistart version of our heuristic, where the number of starts is fixed after experimentation at 5. Table 3 summarizes the obtained results. In this table, for each method, we provide:

- $\#BEST$: The number of times it yields the best known solution.
- GAP: The gap to the best solution. The GAP for each instance is calculated using the following expression:

$$GAP = \frac{Z_{best} - Z_{max}}{Z} \times 100 \tag{22}$$

where Z_{best} is the best solution found by the heuristic and Z_{max} is the best solution in the literature, including our method.

- CPU: The total CPU time.

Table 3: Performance of our heuristic

Class	COP-TABU-*Basic*			COP-TABU-*Multistart*			COP-TABU-*Reactive*			Hybrid Heuristic		
	$\#BEST$	GAP	CPU	$\#BEST$	GAP	CPU	$\#BEST$	GAP	CPU	$\#BEST$	GAP	CPU
$dantzig42$	16	0	13.27	16	0	17.77	16	0	38.95	16	0	9.12
$swiss42$	13	0.719	15.38	14	0.281	23.09	15	0.013	31.93	16	0	6.70
$att48$	16	0	18.24	16	0	26.08	15	0.062	38.76	16	0	22.42
$gr48$	11	5.709	13.74	12	3.184	26.02	16	0	37.96	16	0	9.54
$hk48$	15	1.250	20.58	16	0	30.76	15	0.315	37.68	16	0	12.87
$eil51$	11	2.181	15.92	11	2.181	24.46	15	0.242	36.83	14	0.326	10.61
$berlin52$	15	0.548	38.88	15	0.120	53.39	15	0.120	60.41	16	0	29.50
$brazil58$	13	0.573	58.99	14	0.115	75.72	16	0	83.97	16	0	49.60
$st70$	11	1.303	23.18	11	1.012	38.95	12	0.639	48.01	16	0	18.00
$eil76$	9	6.407	24.5	10	4.050	33.74	15	0.125	45.84	16	0	12.18
$pr76$	11	1.014	21.4	13	0.105	30.88	15	0.009	54.76	16	0	29.94
$gr96$	12	0.612	44.07	13	0.116	51.35	14	0.025	68.19	16	0	31.46
$rat99$	12	1.752	32.99	12	0.127	52.03	15	0.034	63.65	15	0.151	34.02
$kroA100$	11	6.013	44.65	14	0.123	50.98	14	0.429	52.62	15	0.082	22.03
$kroB100$	15	0.714	47.96	16	0	58.94	16	0	62.2	16	0	21.02
$kroC100$	10	3.687	37.55	15	0.269	48.74	14	0.452	59.42	16	0	23.73
$kroD100$	10	1.879	36.85	11	1.247	56.7	13	0.520	69.57	16	0	32.41
$kroE100$	12	2.889	46.59	12	1.374	48.83	14	0.270	62.77	16	0	21.97
$rd100$	12	1.431	36.51	13	1.030	47.81	15	0.568	82.29	16	0	28.31
$eil101$	7	2.495	32.97	12	0.729	44.62	16	0	79	16	0	32.81
$lin105$	11	1.393	36.06	13	0.461	52.48	14	0.348	105.21	16	0	68.60
$pr107$	13	6.350	72.19	15	0.203	86.35	15	0.160	135.39	16	0	204.68
$gr120$	10	2.917	50.87	11	2.856	66.36	14	0.185	105.25	15	0.054	54.16
$pr124$	14	1.180	80.33	16	0	88.26	16	0	150.15	16	0	87.29
$bier127$	12	0.873	63.05	14	0.108	94.57	15	0.005	149.64	16	0	69.41
$ch130$	7	4.016	49.79	9	2.949	64.58	12	1.376	106.57	16	0	44.22
$pr136$	12	1.588	59.86	14	0.949	71.37	15	0.694	121.5	16	0	57.12
$gr137$	15	0.156	82.07	16	0	104.45	16	0	181.54	16	0	62.47
$pr144$	16	0	168.25	16	0	175.28	16	0	247.29	16	0	124.72
$ch150$	8	2.684	34.19	8	2.543	53.97	14	0.554	101.37	16	0	59.81
$kroA150$	9	1.002	36.84	13	0.228	50.6	14	0.074	102.11	15	0.046	54.30
$kroB150$	8	2.456	40.06	10	2.127	56.69	14	0.621	107.93	16	0	51.66
$pr152$	15	0.545	120.08	16	0	164.81	16	0	248.1	16	0	126.88
$u159$	6	3.300	113.36	9	2.373	125.51	8	1.447	184.68	15	0.281	162.81
$si175$	16	0	47.69	16	0	63.36	16	0	126.8	16	0	1022.91
$brg180$	12	0.656	54.29	13	0.578	72.18	15	0.091	127.74	16	0	605.49
$rat195$	12	0.531	68.18	10	0.209	78.52	14	0.401	172	16	0	162.70
$d198$	15	0.062	172.56	16	0	217.29	16	0	368.98	16	0	147.69
$kroA200$	11	1.130	55.09	12	1.093	76.17	14	1.052	139.43	16	0	100.08
$kroB200$	8	2.610	71.45	10	1.978	87.73	13	0.129	142.43	15	0.034	103.26
$gr202$	11	1.256	88.17	12	1.001	121.27	16	0	236.24	16	0	147.79
$ts225$	12	0.259	162.94	12	0.158	189.13	15	0.019	234.81	16	0	204.00
$tsp225$	9	1.583	87.78	9	0.495	102.99	11	0.142	180.26	16	0	191.03
$pr226$	12	0.872	244.84	12	0.787	268.33	15	0.042	331.1	16	0	314.10
$gr229$	15	0.023	109.99	15	0.023	121.07	15	0.023	170.85	16	0	96.69
$gil262$	6	8.133	57.09	6	4.324	84.2	10	2.469	135.48	14	0.051	183.41
$pr264$	11	4.230	151.96	10	4.243	208.51	14	0.323	304.7	16	0	256.35
$a280$	11	0.159	99.98	12	0.156	150.54	10	0.255	191.39	14	0.402	690.45
$pr299$	11	0.782	105.14	11	0.774	125.98	13	0.298	205.23	16	0	515.12
$lin318$	8	0.997	247.01	9	0.858	260.81	11	0.480	311.25	15	0.020	733.50
$rd400$	11	0.954	100.44	12	0.375	147.13	13	0.351	203.08	15	0.542	557.71
$fl417$	11	1.055	518.97	12	0.397	577.53	13	0.079	708.57	16	0	672.82
$gr431$	12	0.788	236.75	15	0.009	252.35	16	0	280.53	16	0	429.84
$pr439$	11	0.685	180.16	13	0.074	221.23	14	0.058	324.17	15	0.002	531.14
$pcb442$	12	0.295	151.28	11	0.512	199.82	13	0.495	274.18	14	0.244	840.38

continued on next page

Table 3 – continued from previous page

Class	COP-TABU-*Basic*			COP-TABU-*Multistart*			COP-TABU-*Reactive*			Hybrid Heuristic		
	#BEST	*GAP*	*CPU*	*#BEST*	*GAP*	*CPU*	*#BEST*	*GAP*	*CPU*	*#BEST*	*GAP*	*CPU*
*d*493	7	1.157	418.35	9	1.208	419.66	12	1.051	515.84	16	0	949.26
*att*532	16	1.768	1707.99	19	1.461	2180.37	25	0.938	2244.39	23	0.273	3590.19
Total	658	1.473	153.719	722	0.816	174.37	819	0.302	223.88	902	0.045	301.73

On the basis of Table 3, we conclude that our heuristic algorithm outperforms all COP-TABU versions in terms of total number of best solutions and the gap to the best. It delivers 38 new BKS to reach 902 BKS. On the other hand the three tabu versions heuristics yield 658, 722 and 819 BKS. Moreover, our heuristic exhibits a relatively small gap to the best of 0.045 compared to 0.302 for COP-TABU (*Reactive*). Finally, we observe that our method substantially improves the results of several classes of instances namely: *ch*150, *u*159, *tsp*225, *gil*262 and *d*493.

5.2. Set B: multiple vehicles problem set

5.2.1. Description of the instances

Since there are no instances for CluTOP in the literature, we generated a new set of instances to evaluate the effectiveness of our methods in the case of multiple vehicles. We proceeded in the same way as in Chen et al. [5]. The authors generated new instances of the TOP from instances of the OP by dividing the T_{max} by the number of vehicles in such a way that each vehicle has a time limit equal to $\frac{T_{max}}{m}$. As a result, from each benchmark instance in [1], we derived two instances with a number of vehicles two or three, knowing that T_{max} values in [1] were calculated as $\theta * TSP$, where θ takes two values: $\frac{1}{2}$ and $\frac{3}{4}$. Hence, the total number of the new instances is 924 for each number of vehicles.

5.2.2. Performance of the exact method

Table 4 shows the performance of the cutting plane when considering one, two and three vehicles. From Table 4, we observe that the instances become more difficult to solve when considering multiple vehicles. For instance, in the case of a single vehicle, all of the instances with less than 100 customers were solved to optimality by the cutting plane, whereas for two and three vehicles, the number of solved instances per class is remarkably low. For example, the instances of class *rat*99 were all solved to optimality in the case of a single vehicle, but only eight of them were solved when the number of vehicles is equal to two or three.

Like in Section 5.1, performance evaluation of the cutting plane in the case of multiple vehicles was restricted to instances with up to 318 vertices and 25 clusters. Therefore, 800 instances were considered for each class. In this section, we attempted to study the impact of the number of vehicles on the behavior of the exact method.

Table 4 shows the performance of the cutting plane when considering one, two and three vehicles. From Table 4, we observe that the instances become

more difficult to solve when considering multiple vehicles. For instance, in the case of a single vehicle, all of the instances with less than 100 customers were solved to optimality by the cutting plane, whereas for two and three vehicles, the number of solved instances per class is remarkably low. For example, the instances of class *rat*99 were all solved to optimality in the case of a single vehicle, but only eight of them were solved when the number of vehicles is equal to two or three.

Our exact method exhibits a percentage gap of 26% for two vehicles, 30.1% for three vehicles compared to 4.5% with a single vehicle. Moreover, we observe the same trend for the number of solved instances and computational times. Regarding the case of three vehicles, the cutting plane succeeded to solve more instances than the case with two vehicles. This is mainly due to the fact that the routing sub-problem becomes relatively easier since the T_{max} is divided by the number of vehicles. Furthermore, many instances have an objective value equal to zero or they at least contain the profit of a very small number of clusters.

Table 4: Performance of the cutting plane with respect to the number vehicles

Class	Single vehicle			Two vehicles			Three vehicles		
	#OPT	*OptGap*	*CPU*	*#OPT*	*OptGap*	*CPU*	*#OPT*	*OptGap*	*CPU*
*dantzig*42	16	0	2.90	16	0	60.25	16	0	0.4
*swiss*42	16	0	4.08	16	0	71.8	16	0	129.3
*att*48	16	0	34.80	16	0	377.6	16	0	10.29
*gr*48	16	0	12.46	11	0.060	1323.46	16	0	1.13
*hk*48	16	0	12.42	12	0.065	1208.14	16	0	2.23
*eil*51	16	0	16.56	15	0.004	702.63	10	0.097	1619.76
*berlin*52	16	0	9.96	16	0	700.98	16	0	703.23
*brazil*58	16	0	26.67	15	0.005	793.99	16	0	165.53
*st*70	16	0	95.71	8	0.214	1800.12	16	0	59.91
*eil*76	16	0	25.83	8	0.069	2271.91	8	0.241	1909.14
*pr*76	16	0	196.43	10	0.058	1853.91	8	0.128	1802.68
*gr*96	16	0	37.23	7	0.079	2353.22	13	0.030	1270.34
*rat*99	16	0	159.30	8	0.114	1991.96	8	0.140	1805.56
*kroA*100	16	0	240.96	8	0.280	1808.33	12	0.061	941.89
*kroB*100	16	0	272.45	8	0.255	1812.47	15	0.012	458.12
*kroC*100	15	0.005	746.34	8	0.307	1803.14	14	0.023	550.91
*kroD*100	16	0	581.61	8	0.276	1803.88	14	0.090	488.04
*kroE*100	16	0	179.06	8	0.394	1800.26	16	0	0.14
*rd*100	16	0	225.11	8	0.207	1832.97	12	0.173	995.38
*eil*101	16	0	115.96	5	0.123	3198.43	0	0.386	3600.03
*lin*105	16	0	72.66	10	0.038	1737.53	8	0.131	1800.17
*pr*107	15	0.019	332.52	16	0.000	6.81	16	0	0.15
*gr*120	15	0.006	688.78	2	0.341	3169.59	8	0.367	1800.31
*pr*124	16	0	100.51	14	0.016	1132.63	13	0.018	1144.92
*bier*127	16	0	92.16	3	0.170	3257.18	0	0.289	3600.16
*ch*130	15	0.0042	767.34	4	0.328	2823.47	6	0.348	2594.12
*pr*136	16	0	326.29	1	0.356	3544.06	8	0.309	1800.65
*gr*137	16	0	149.90	6	0.170	2587.61	9	0.130	1784.16
*pr*144	16	0	135.73	8	0.127	1843.67	14	0.041	934.23
*ch*150	9	0.0584	2098.01	4	0.414	2758.08	8	0.453	1800.74
*kroA*150	10	0.0583	1713.36	4	0.381	2722.32	8	0.547	1807.96
*kroB*150	11	0.0563	1913.78	6	0.396	2569.58	8	0.539	1800.4
*pr*152	16	0	559.19	8	0.205	1800.49	16	0	0.49
*u*159	14	0.0109	1185.79	0	0.264	3600.14	3	0.218	2999.66
*si*175	16	0	315.50	2	0.095	3187.11	0	0.192	3600.07
*brg*180	16	0	155.60	10	0.024	2589.05	2	0.149	3328.89
*rat*195	10	0.0447	1757.66	0	0.334	3600.18	0	0.447	3600.15

continued on next page

Class	Single vehicle			Two Vehicles			3 Vehicles		
	$\#OPT$	$OptGap$	CPU	$\#OPT$	$OptGap$	CPU	$\#OPT$	$OptGap$	CPU
$d198$	12	0.0332	1431.59	8	0.191	1801.94	16	0	2.27
$kroA200$	6	0.1473	2562.21	2	0.594	3171.66	8	0.583	1802.31
$kroB200$	6	0.1767	2550.03	2	0.622	3164.14	8	0.601	1802.45
$gr202$	16	0	557.13	0	0.397	3600.87	2	0.351	3436.75
$ts225$	0	0.2729	3600.73	0	0.742	3601.16	8	0.496	1908.84
$tsp225$	10	0.0681	2461.35	0	0.499	3600.76	0	0.607	3600.47
$pr226$	5	0.1441	3066.90	8	0.179	2014.41	16	0	210.16
$gr229$	16	0	1063.57	0	0.605	3601.41	0	0.686	3602.57
$gil262$	4	0.1986	3084.40	0	0.827	3600.66	8	0.711	1801.03
$pr264$	2	0.3496	3155.57	8	0.281	1817.96	10	0.049	1358.66
$a280$	2	0.1768	3357.49	0	0.461	3600.39	0	0.518	3600.16
$pr299$	1	0.2837	3449.92	0	0.509	3600.8	2	0.402	3151.43
$lin318$	1	0.1547	3430.73	0	0.517	3600.81	0	0.746	3600.41
$Total$	643	0.0454	982.64	337	0.252	2265.52	463	0.226	1615.77

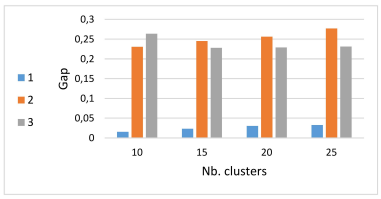

Figure 2: Optimality Gap with respect to the number of clusters and the number of vehicles.

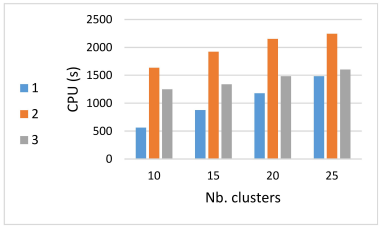

Figure 5: Computational time with respect to the number of clusters and the number of vehicles.

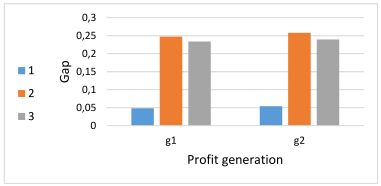

Figure 3: Optimality Gap with respect to the number of profit generation and the number of vehicles.

Figure 6: Computational time with respect to profit generation and the number of vehicles.

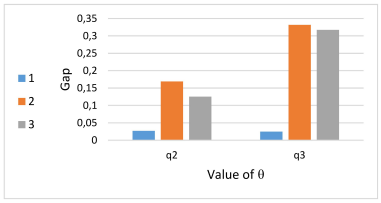

Figure 4: Optimality Gap with respect to θ and the number of vehicles.

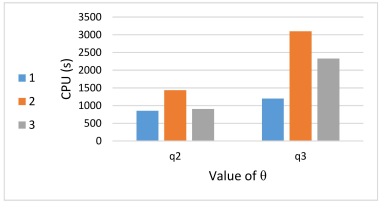

Figure 7: Computational time with respect to θ and the number of vehicles.

We investigate now the impact of the use of multiple vehicles on the performance of the cutting plane. Figures 2 to 7 depict some performance mea-

surements of the cutting plane with respect to the number of vehicles. We investigated the variations of the optimality gap and computational time with respect to the number of clusters, the profit generation pattern and the value of θ. Globally, we note that instances with several vehicles become more difficult than the case with a single vehicle.

5.2.3. Performance of the heuristic method

In this last section, we discuss the performance of our proposed heuristic in the case of multiple vehicles. In Table 5, we provide the following measurements for our heuristic:

- $UBGap$: Average deviation gap with respect to an upper bound;
- $OptGap$: Average deviation gap with respect to the optimal solution value;
- $\#Opt$: Number of times it yields the optimal solution;
- CPU: Computational times;

The results depicted in Table 5 confirm the robustness of our heuristic method in the case of multiple vehicles. Computational times remain relatively stable and very close to the instances with a single vehicle. We observe that $UBGap$ drastically increases in the case of multiple vehicles compared to a single vehicle. This is mainly due to the poorness of the upper bounds rather than to the performance of the heuristic since the gap to the optimal solution ($OptGap$) is 0.02. In addition, the heuristic method succeeds in finding almost all the optimal solutions found by the exact method: 328/337 with two vehicles, 460/463 with three vehicles and 673/679 in the case of a single vehicle.

Table 5: Multiple vehicles

	$UBGap(\%)$	$OptGap(\%)$	$\#Opt$	$CPU(s)$
1 vehicle	0.888	0.02	673/679	301.73
2 vehicles	11.201	0.141	328/337	390.07
3 vehicles	11.6	0.258	460/463	310.411

Figures 8 to 10 depict the performance of the heuristic with respect to the number of vehicles and the number of clusters. According to fig. 8, the gap to the upper bound does not show any clear behavior since the gap remains relatively high regardless of the number of clusters for two and three vehicles. This is mainly due to the poorness of the upper bounds as described earlier. Regarding computational times, we can observe that the number of vehicles does not affect the behavior of the heuristic for instances with up to 25 clusters (fig. 9). Interestingly, for instances with 532 vertices (att532), we notice that the behavior of the heuristic incurs some changes especially for 50, 75 and 100 clusters, as shown in fig. 10. Computational times of the heuristic tend to

stabilize in the case of two and three vehicles, but continues to increase in the case of a single vehicle.

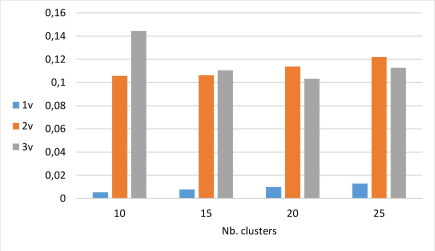

Figure 8: Average gap to the best upper bound.

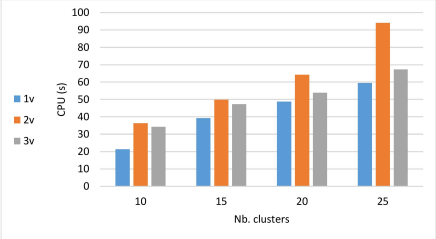

Figure 9: Computational times with respect to the number of clusters.

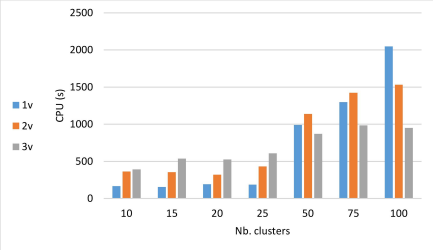

Figure 10: Computational times with respect to the number of clusters on instances with 532 vertices.

6. Conclusion and future work

In this paper, we introduced a new variant of the TOP, called the Clustered Team Orienteering Problem (CluTOP). In the CluTOP, customers are grouped into subsets called clusters, to which we assign profits representing the value of service. the CluTOP also generalizes the Clustered Orienteering Problem (COP) where only a single vehicle is used. We proposed an exact and a heuristic methods to solve the CluTOP. Results for a single vehicle show the competitiveness of both of our methods compared to the literature, with new solved instances by the exact methods and new best known solutions by the

heuristic one. In future studies, our aim is to propose different extensions for the CluTOP by considering new additional constraints like time windows or vehicle capacity.

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). Thanks are due to anonymous referees for their valuable comments that helped improving this paper.

References

[1] Enrico Angelelli, Claudia Archetti, and Michele Vindigni. The clustered orienteering problem. European Journal of Operational Research, 238(2):404–414, 2014.

[2] Claudia Archetti, Francesco Carrabs, and Raffaele Cerulli. The set orienteering problem. European Journal of Operational Research, 267(1):264–272, 2018.

[3] Claudia Archetti, M Grazia Speranza, and Daniele Vigo. Vehicle routing problems with profits. Vehicle Routing: Problems, Methods, and Applications, 18:273, 2014.

[4] John E Beasley. Route first-cluster second methods for vehicle routing. Omega, 11(4):403–408, 1983.

[5] I-Ming Chao, Bruce L Golden, and Edward A Wasil. The team orienteering problem. European journal of operational research, 88(3):464–474, 1996.

[6] Duc-Cuong Dang and Aziz Moukrim. Subgraph extraction and metaheuristics for the maximum clique problem. Journal of Heuristics, 18(5):767–794, 2012.

[7] Christof Defryn and Kenneth Sörensen. A fast two-level variable neighborhood search for the clustered vehicle routing problem. Computers & Operations Research, 83:78–94, 2017.

[8] Racha El-Hajj, Duc-Cuong Dang, and Aziz Moukrim. Solving the team orienteering problem with cutting planes. Computers & Operations Research, 74:21–30, 2016.

[9] Dominique Feillet, Pierre Dejax, and Michel Gendreau. Traveling salesman problems with profits. Transportation science, 39(2):188–205, 2005.

[10] 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.

[11] Matteo Fischetti, Juan José Salazar González, and Paolo Toth. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. Operations Research, 45(3):378–394, 1997.

[12] Aldy Gunawan, Hoong Chuin Lau, and Pieter Vansteenwegen. Orienteering problem: A survey of recent variants, solution approaches and applications. European Journal of Operational Research, 255(2):315–332, 2016.

[13] Kees Jongens and Ton Volgenant. The symmetric clustered traveling salesman problem. European Journal of Operational Research, 19(1):68–75, 1985.

[14] Shen Lin and Brian Kernighan. An effective heuristic algorithm for the traveling-salesman problem. Operations research, 21(2):498–516, 1973.

[15] Patric RJ Östergård. A fast algorithm for the maximum clique problem. Discrete Applied Mathematics, 120(1-3):197–207, 2002.

[16] Ulrich Pferschy and Rostislav Staněk. Generating subtour elimination constraints for the tsp from pure integer solutions. Central European Journal of Operations Research, 25(1):231–260, 2017.

[17] David Pisinger and Stefan Ropke. Large neighborhood search. In Handbook of metaheuristics, pages 399–419. Springer, 2010.

[18] 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.

[19] Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. The orienteering problem: A survey. European Journal of Operational Research, 209(1):1–10, 2011.

[20] Thibaut Vidal. Split algorithm in o (n) for the capacitated vehicle routing problem. Computers & Operations Research, 69:40–47, 2016.

[21] Ala-Eddine Yahiaoui, Aziz Moukrim, and Mehdi Serairi. Hybrid heuristic for the clustered orienteering problem. In International Conference on Computational Logistics, pages 19–33. Springer, 2017.