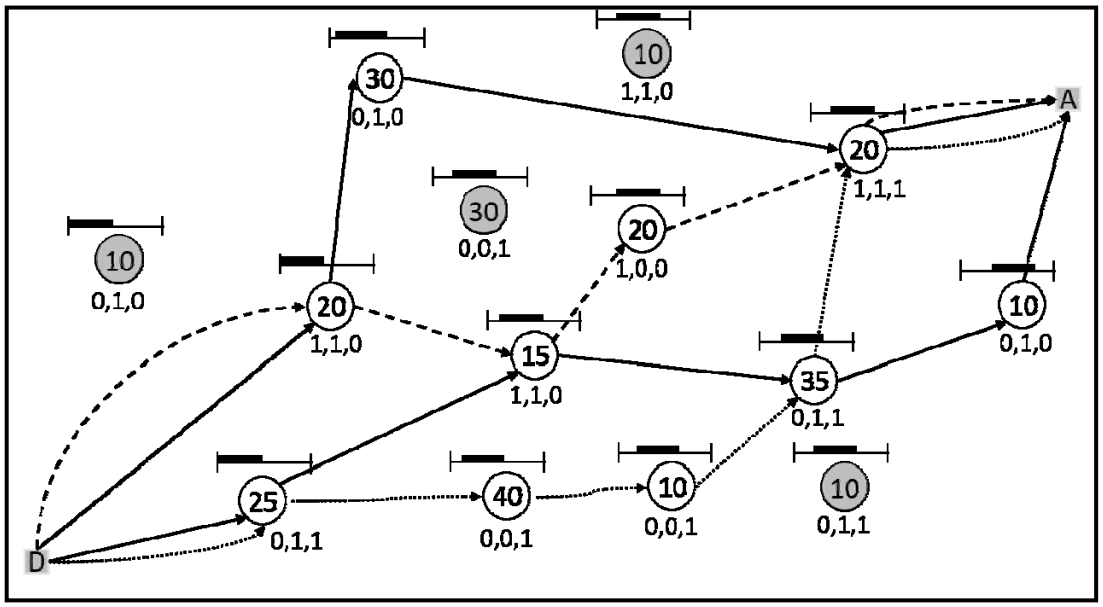
Selective vehicle routing problem : cluster and synchronization constraints

Ala-Eddine Yahiaoui

Par **Ala-Eddine YAHIAOUI**

Selective vehicle routing problem: cluster and synchronization constraints

Thèse présentée
pour l'obtention du grade
de Docteur de l'UTC



Soutenue le 11 décembre 2018
Spécialité : Technologies de l'Information et des Systèmes : Unité de recherche Heudyasic (UMR-7253)

D2449

A thesis presented for the degree of Doctor

UNIVERSITÉ DE TECHNOLOGIE DE COMPIÈGNE

Département Génie Informatique HeuDiaSyC, UMR CNRS 7253

Discipline: TIS

Spécialité : Technologies de l'Information et des Systèmes

Selective vehicle routing problem: cluster and synchronization constraints

by **Yahiaoui Ala-Eddine**

Jury

Jacques Carlier	Université de Technologie de Compiègne	Examinator
Haoxun Chen	Université de Technologie de Troyes	Reporter
Gilles Goncalves	Université d'Artois	Reporter
Slim Hammadi	École Centrale de Lille	Examinator
Aziz Moukrim	Université de Technologie de Compiègne	Supervisor
Wahiba Ramdane-Cherif	École des Mines de Nancy	Examinator
Mehdi Serairi	Université de Technologie de Compiègne	Supervisor

Thesis defense: 11 December 2018

Avant propos

Cette thèse a été réalisée dans le laboratoire Heudiasyc, au sein de l'équipe SCOP (ex. RO). La thèse a été cofinancée par la région Hauts-de-France et le Fond Européen de Développement Économique et Régional (FEDER).
Je voudrais d'abord exprimer mes sincères remerciements et gratitudes à mes directeurs de thèse, Pr. Aziz MOUKRIM et Dr. Mehdi SERAIRI pour leurs accompagnements et leurs soutiens tout au long de ce travail. Je leur suis très reconnaissant de m'avoir accordé leur confiance pour mener ce travail tout en étant présents pour m'aider.
Je remercie M. Gilles GONCALVES, Professeur à l'Université d'Artois, et M. Haoxun CHEN, Professeur à l'Université de Technologie de Troyes, pour le temps qu'ils ont consacré à relire cette thèse en tant que rapporteurs et pour les remarques pertinentes vis-à-vis de mon travail. J'adresse également mes remerciements au M. Jacques CARLIER, Professeur à l'Université de Technologie de Compiègne, pour avoir présidé mon jury de thèse.
Je remercie M. Slim HAMMADI, Professeur à l'Université de Lille, et Mme. Wahiba CHERIF-KHETTAF, Maître de Conférence à l'École des Mines de Nancy, qui m'ont fait l'honneur d'avoir accepté de participer au jury de ma thèse.
Je voudrais également remercier particulièrement mes parents, mes frères et mes sœurs pour leur soutien permanent et inconditionnel. Les mots ne peuvent exprimer à quel point je suis reconnaissant à mes parents, leurs prières et invocations m'ont soutenu durant tout mon parcours jusqu'à aujourd'hui.
Finalement, je tiens à remercier mes collègues du laboratoire HEUDIASYC pour leur accueil chaleureux et l'ambiance qui a régné au sein du laboratoire. Je tiens à remercier particulièrement mon ami Djamel-Eddine KOUICEM pour les nombreux débats scientifiques et intellectuels que nous avons eus ensemble et qui ont constitué un défouloir pendant les moments les plus difficiles de la thèse.

Table des matières

Liste des figures

Liste des tableaux

Liste des algorithmes

Publications

In proceedings

[1] AE. Yahiaoui, A. Moukrim and M. Serairi. A Hybrid Heuristic for the Clustered Orienteering Problem. In : Proc. of ICCL-8, 2017 (Southhampton, England), series : Lecture Notes in Computer Science, vol 10572, pp 19-33.

Seminars and communications

[2] AE. Yahiaoui, A. Moukrim and M. Serairi. Split Methods for the Clustered Team Orienteering Problem. International Conference on Operations Research -Analytical Decision Making. Hambourg, 2016.
[3] AE. Yahiaoui, A. Moukrim and M. Serairi. Problème de tournées de véhicules sélectives avec contraintes de clusters. Livre des résumés de la ROADEF 2018 (Lorient, France).

Journal articles

[4] AE. Yahiaoui, A. Moukrim and M. Serairi. The Team Orienteering Problem with Cluster Constraint. *Submitted to Computers and Operations Research*

In preparation

[5] AE. Yahiaou, A. Moukrim and M. Serairi. GRASP×ILS with Constraint Programming for the Synchronized Team Orienteering Problem with Time Windows. *To be submitted*

Résumé

Le problème de tournées de véhicules (Vehicle Routing Problem - VRP) est un problème d'optimisation combinatoire utilisé généralement pour modéliser et résoudre des différents problèmes rencontrés dans les systèmes logistiques et de transport. Dans cette thèse, nous nous sommes intéressés à l'étude et la résolution d'une classe de problèmes du VRP appelée les problèmes de courses d'orientation (Team Orienteering Problem - TOP). Dans cette catégorie de problèmes, il est a priori impossible de visiter tous les clients en raison de ressources limitées. On associe plutôt un profit à chaque client qui représente sa valeur. Ce profit est collecté lorsque le client est visité par l'un des véhicules disponibles. L'objectif est donc de sélectionner un sous ensemble de clients à servir tout en maximisant le profit total collecté. Dans un premier temps, nous avons introduit une nouvelle généralisation pour le TOP que nous avons appelée le Clustered TOP ou CluTOP. Dans cette variante, les clients sont regroupés en sous-ensembles appelés *clusters* auxquels nous associons des profits. Pour résoudre cette variante, nous avons proposé un schéma exact basé sur l'approche des plans sécants avec des inégalités valides supplémentaires et des pré-traitements. Nous avons également conçu une méthode heuristique basée sur l'approche *order first-cluster second*. Cette heuristique hybride combine une heuristique de type Adaptive Large Neighborhood Search qui explore l'espace des solutions et une procédure de découpage qui explore l'espace de recherche des tours géants. De plus, la procédure de découpage est renforcée par une recherche locale afin de mieux explorer l'espace de recherche. Le deuxième problème traité dans ce travail s'appelle le Synchronized Team Orienteering Problem with Time Windows (STOPTW). Cette variante avait été initialement proposée afin de modéliser des scénarios liés à la protection des infrastructures stratégiques menacées par l'avancée des feux de forêts. En plus des contraintes de fenêtres de temps et des visites synchronisées, cette variante considère le cas d'une flotte de véhicules hétérogène. Pour résoudre ce problème, nous avons proposé une méthode heuristique basée sur l'approche GRASP×ILS qui est parvenue à dominer la seule approche existante dans la littérature. La dernière variante du TOP abordée dans cette thèse s'appelle le Set Orienteering Problem (SOP). Les clients dans cette variante sont regroupés en sous-ensembles appelés *clusters*. Un profit est associé à chaque groupe qui n'est

obtenu que si au moins un client est desservi par le véhicule disponible. Nous avons proposé une méthode de coupes avec deux procédures de séparation pour séparer les contraintes d'élimination des sous-tours. Nous avons également proposé un algorithme Memetique avec une procédure de découpage optimale calculée à l'aide de la programmation dynamique.

Mots Clés : Problèmes de Tournées de Véhicules Sélectives, Cluster, Synchronisation, Méthode de découpage, Méthode de plans sécants.

Abstract

The Vehicle Routing Problem (VRP) is a family of Combinatorial Optimization Problems generally used to solve different issues related to transportation systems and logistics. In this thesis, we focused our attention on a variant of the VRP called the Team Orienteering Problem (TOP). In this family of problems, it is *a priory* impossible to visit all the customers due to travel time limitation on vehicles. Instead, a profit is associated with each customer to represent its value and it is collected once the customer is visited by one of the available vehicles. The objective function is then to maximize the total collected profit with respect to the maximum travel time. Firstly, we introduced a new generalization for the TOP that we called the Clustered TOP (CluTOP). In this variant, the customers are grouped into subsets called clusters to which we associate profits. To solve this variant, we proposed an exact scheme based on the cutting plane approach with additional valid inequalities and pre-processing techniques. We also designed a heuristic method based on the order first-cluster second approach for the CluTOP. This Hybrid Heuristic combines between an ANLS heuristic that explores the solutions space and a splitting procedure that explores the giant tours search space. In addition, the splitting procedure is enhanced by local search procedure in order to enhance its coverage of search space. The second problem treated in this work is called the Synchronized Team Orienteering Problem with Time Windows (STOPTW). This variant was initially proposed in order to model scenarios related to asset protection during escaped wildfires. It considers the case of a heterogeneous fleet of vehicles along with time windows and synchronized visits. To solve this problem, we proposed a heuristic method based on the GRASP×ILS approach that led to a very outstanding results compared to the literature. The last variant of the TOP tackled in this thesis called the Set Orienteering Problem (SOP). Customers in this variant are grouped into subsets called clusters. Each cluster is associated with a profit which is gained if at least one customer is served by the single available vehicle. We proposed a Branch-and-Cut with two separation procedures to separate subtours elimination constraints. We also proposed a Memetic Algorithm with an optimal splitting procedure based on dynamic programming.

Key Words : Team Orienteering Problem, Cluster , Cutting Planes, Synchro-

nization, Splitting procedure.

Introduction

Transportation and logistics is a major component of the economy of today's society. Statistics estimate at 5% to 20% the part of transportation costs in the GDP of developed countries. This percentage tends to be higher in the next few years due to the increasing number of businesses based on the transportation of persons and goods, like e-commerce, home health care, etc. As a result, optimizing the operations costs and resource consumption becomes a major issue in order to guarantee a long term growth of the whole economy. Combinatorial Optimization is an academic field that provides a set of tools to help designers optimizing the consumption of resources in transportation and logistics systems. The main idea is to design efficient algorithmic procedures for planning the distribution process. Any improvement achieved by these procedures in a computer-simulation-based environment can lead to a substantial reduction of costs in real-life applications.

Vehicle Routing Problems (VRPs) is a family of Combinatorial Optimization Problems generally used to solve different issues in transportation systems. Since the first definition of the VRP by Dantzig and Ramser in [Dantzig and Ramser, 1959], several variants have been proposed throughout the years in order to deal with various practical applications and challenges related to transportation. This includes the original VRP with additional characteristics used to model customers' preferences and requirements like time windows, pickup and delivery services, as well as constraints imposed on the vehicles like limited travel times, multiple depots, multiple periods, etc.

Vehicle routing problems or Combinatorial Optimization Problems in general can naively be solved by exhaustive enumeration of all feasible solutions. However, this approach requires exponential computational times. The challenge in Combinatorial Optimization consists in designing intelligent techniques to find the best solutions without enumerating all the search space. Such approaches are easy to find for some problems, whereas for the vast majority of them it is difficult to find a good one. Consequently, optimization problems are grouped into two formal classes, those which are easy to solve and those which are not. In the context of this thesis, we are interested in resolution approaches for hard optimization problems. We distinguish two categories of approaches : exact methods that aim at finding the best

solution, whereas heuristic methods try to find good quality solutions at reasonable computational times.

In this thesis, we propose to tackle a specific class of VRPs called the Team Orienteering Problem (TOP) and its variants. This class of problems is characterized by some additional constraints and resource limitations, which make it impossible to serve all the customers. In the TOP for example, a limited fleet of vehicles is available to visit the potential set of customers. Each vehicle must start from a departure and ends in the arrival depot without exceeding a travel time limit. In order to discriminate between customers, each one is assigned a value called *profit*. The objective is therefore to select a subset of customers to serve in such a way that the total collected profit is maximized while respecting the resource limitation.

We focus in this thesis on some variants of the TOP with some additional constraints derived from real-life applications. A main characteristic of the studied problems is that profits are associated with subsets of customers rather than individual customers. In the Clustered Team Orienteering Problem (CluTOP) for example, subsets are called *clusters*, and in order to collect the profit of a given cluster, vehicles must visit all of its customers. Another variant tackled in this thesis is the Set Orienteering Problem (SOP), in which the profit of a cluster is gained if at least one of its customers is served. Finally, in the Synchronized Team Orienteering Problem with Time Windows (STOPTW), one customer may require to be visited by multiple vehicles at the same time in order to gain its profit. Time windows are used here to model the availability of customers to receive the service performed by the vehicles.

Several efficient methods, exact and approximate, were proposed to solve these problems. In chapter 1, we provide the reader with an overview of the combinatorial optimization field. We start by giving a brief and concise description of the main concepts and terminologies used in combinatorial optimization. We also present a number of exact and approximate solution methods generally used to tackle vehicle routing problems. We describe at the end the most studied variants of vehicle routing problems as well as the selective case.

In chapter 2, we introduce the Clustered Team Orienteering Problem (CluTOP). This new problem is in fact a generalization to multiple vehicles of the Clustered Orienteering Problem (COP) proposed in [Angelelli et al., 2014]. After reviewing the literature and introducing a mathematical model for the CluTOP, we propose an efficient exact method based on cutting planes approach. In this method, we first remove subtour elimination constraints from the original model yielding to a new one with a polynomial number of variables and constraints. The new model is then solved and the needed subour elimination constraints are added progressively. The

model is enhanced by a pre-processing procedure deduced after computing cliques on particular graphs that represent incompatibility between customers or clusters. Furthermore, valid inequalities including symmetry breaking cuts and bounds on profits and the number of clusters are added to the model. The exact method shows very good performance in the case of single vehicle by finding the optimal solution for a large number of instances still unsolved in the literature [Angelelli et al., 2014].

In chapter 3, we propose a hybrid heuristic method to solve the CluTOP. Only one method was proposed for the problem in the case of a single vehicle [Angelelli et al., 2014]. Our method explores both solution and giant tour search spaces. The method incorporates an Adaptive Large Neighborhood Search heuristic used to explore solutions space. An optimal splitting procedure is proposed in order to extract solutions from giant tours. It is based on a branch-and-bound scheme enhanced by a knapsack-based upper bound to fathom inferior nodes. Our hybrid heuristic outperforms the existing method and achieves to improve strictly a large number of instances.

Then, in chapter 4, we explore a recent variant of the TOP called the Synchronized Team Orienteering Problem with Time Windows (STOPTW). This problem is addressed as part of the European project GEOSAFE carried out in collaboration with Australia. The STOPTW is used to model the asset protection problem during escaped wildfires [Roozbeh et al., 2018], where customers represent vital assets endangered by the progression of wildfires. In this problem, protection activities are carried out by firefighting teams that use a limited fleet of heterogeneous vehicles. Each asset has a time window which estimates the time to impact of fire fronts. Each asset requires a specific number of vehicles of different types in order to ensure its protection. In addition, the required vehicles must visit the asset in a synchronized manner, i.e. simultaneously within the corresponding time window. The objective in STOPTW is to maximize the total collected profit with respect to synchronization and time window constraints. After reviewing some literature about vehicle routing problems applications in disaster management, we propose a new mathematical formulation for the problem along with a constraint programming model. We then present a heuristic method of type Greedy Randomized Adaptive Search Procedure (GRASP) couple with an Iterated Local Search and enhanced by CP-based insertion module. Also, a post optimization phase is performed using a set cover formulation that extracts the best solution from a pool of routes generated by the GRASP. The results achieved by our method prove its efficiency. It succeeds to dominate the literature on all the benchmark instances.

The last variant of the TOP tackled in the chapter 5 is the Set Orienteering Problem (SOP) proposed in [Archetti et al., 2018]. We present a Memetic Algorithm

to solve the SOP. Our method incorporates an optimal splitting procedure based on dynamic programming and label propagation used to evaluate chromosomes. The split method is enhanced by a Knapsack-based upper bound used to prune unpromising labels at early stages of the method. Moreover, we use a combination of local search techniques and an iterative destructive/constructive heuristic in order to improve offspring after the crossover operation.

This manuscript finishes with a general conclusion. We summarize the major contributions presented in this thesis and highlight the main results achieved, as well as some perspectives on our future work.

Field study

Sommaire

1.1 Introduction

Combinatorial Optimization is at the heart of this work. Many problems of theoretical as well as practical importance consist in searching for the "best" configuration or a set of parameter values to achieve a certain goal. An extensive work has been carried out during the past few decades leading to the definition of a considerable number of such problems. We present in this chapter the optimization basics as well as a quick overview of the objectives of this thesis. We start by giving the main definitions related to Combinatorial Optimization Problems. Then, we provide a list of mathematical tools and algorithms used to solve this kind of problems. At the end of this chapter, we give a general presentation of Team Orienteering Problems with more emphasis on the variants tackled in this manuscript.

1.2 Definitions and terminology

The goal of this section is to introduce the main definitions and the terminology used in the manuscript. A formal definition of *Combinatorial Optimization Problem* ($\mathcal{COP}$) is first presented. Next, we recall the notions of graph, algorithm and complexity.

1.2.1 Combinatorial optimization problem

An optimization problem consists in finding the best solution from a set of available alternatives (called optimum) [Resende and Pardalos, 2008]. An optimal solution is represented by the best variable values with regard to an objective function while respecting a set of constraints. We distinguish two categories of optimization problems : those with *continuous* variables, and those with *discrete* variables. In this thesis, we focus on problems with discrete variables called *combinatorial* [Papadimitriou and Steiglitz, 1998]. A formal definition of Combinatorial Optimization Problem is provided in Definition 3.3.1.

Definition 1.2.1. A combinatorial optimization problem $\Phi = (\Omega, f)$ is defined by :

- A set n of variables $X = \{X_1, \ldots, X_n\}$ where each variable X_i is associated to a domain D_i.
- A set of constraints on the variables.
- An objective function to minimize (or maximize) $f : D_1 \times \ldots \times D_n \to \mathbb{R}$.

The set Ω is called the search space. A feasible solution for Φ is an element $s \in \Omega$ where

$$s = \{v_1, \ldots, v_n | v_i \in D_i \text{ and all the constraints of } \Phi \text{ are satisfied}\}$$

Solving a $\mathcal{COP}$ with a minimization (or a maximization) objective function consists in finding a solution $s^* \in \Omega$ such that $\forall s \in \Omega, f(s^*) \leq f(s)$ (or $f(s^*) \geq f(s)$ with a maximization objective function).

1.2.2 Algorithm complexity

The formal definition of an "algorithm" was introduced by [Turing, 1937]. This definition is founded on the concept of a formal language of an abstract machine called *Turing machine*. Without going into the details of this complex definition, an algorithm A is simply a finite sequence of unambiguous instructions to solve a given problem Φ. For instance, to solve a $\mathcal{COP}$, we need to provide in details all the steps needed to calculate the values of s^*. An algorithm is *exact method* if it returns the optimal solution s^*, while it is *approximate method* if it does not necessarily return s^*, instead it returns a solution s. The quality of such solution is measured using the relative error $|f(s^*) - f(s)|$. Several criteria exist to measure the performance of an algorithm A :

- Runtime : computational time needed by A to find the solution.

- Memory : the size of memory space necessary to run A.
- Quality of the solution : calculated by $|f(s^*) - f(s)|$.
- Robustness : ability of the algorithm to adapt to any small change to the original instance problem.

Since the runtime is tightly related to the characteristics of the machine, the theory of *complexity* has been introduced in order to study the complexity of algorithms regardless the performance of the computing machine. In this theory, each instruction represents an elementary operation (e.g. an addition, a multiplication, an assignment, a test, etc).

Definition 1.2.2. The algorithmic complexity C_A of an algorithm A on a problem Φ is defined as the number of instructions necessary to solve any instance of Φ of size n.

C_A is often expressed asymptotically as a function of n using the big O notation $\mathcal{O}$. A is said to be of complexity $\mathcal{O}(g(n))$ if $\exists M > 0, \exists n_0$ such that $\forall n \geq n_0, C_A \leq Mg(n)$. The most used algorithmic complexities are :

- $\mathcal{O}(1)$: constant and independent of the size of Φ.
- $\mathcal{O}(log\, n)$: logarithmic in the size of Φ.
- $\mathcal{O}(n)$: linear in the size of Φ.
- $\mathcal{O}(n^p)$: (with $p \geq 2$ a constant) polynomial in the size of Φ.
- $\mathcal{O}(b^n)$: (with $b > 1$ a constant) exponential in the size of Φ.

Another case is when complexity is polynomial to the size of the problem and in the same time to the input values, e.g. $g(n) = n^p \times u^q$ with u is a value of the input and q is a constant. The algorithm is called then *pseudo-polynomial*.

1.2.3 NP-Completeness theory

During the past years, algorithm designers have been always seeking efficient algorithms (with polynomial complexity) to solve combinatorial problems. However, many of the confronted problems are inherently intractable and no algorithm could possibly solve them efficiently. Unfortunately, proving inherent intractability of problems can be as hard as finding efficient algorithms to them. In this context, the theory of $\mathcal{NP}$-*Completeness* [Garey and Johnson, 2002] provides many simple

methods for proving that a given problem is as hard as a large number of problems that have been confounding researchers for years. Of course, recognizing that a problem is inherently intractable is just the starting point for algorithm designers, since it provides important information about the most effective solution approach. Before going further, let us first introduce the definition of decision problems.

Definition 1.2.3. A decision problem is a problem where its solution is either YES or NO.

It is noteworthy to mention that for each optimization problem with an objective function f, there is a corresponding decision version. It is formulated as follows : given a real $k \in \mathbb{R}$, does there exist a solution s^* for which $f(s^*) = k$?

Definition 1.2.4. A problem is in class $\mathcal{P}$ (*polynomial time*) if there is an efficient algorithm (with polynomial complexity) that solves it.

Definition 1.2.5. A decision problem is in class $\mathcal{NP}$ (*non-deterministic polynomial time*) if there is an algorithm that can verify in a polynomial time whether a solution is valid.

Definition 1.2.6. A decision problem is in class $\mathcal{NP}$-*Complete* if it is in $\mathcal{NP}$ but not in $\mathcal{P}$, which means that no polynomial algorithm has been found yet to solve it. Moreover, every $\mathcal{NP}$ problem can be reduced into this problem in a polynomial time.

Definition 1.2.7. An optimization problem is in the $\mathcal{NP}$-*Hard* class if its corresponding decision problem is an $\mathcal{NP}$-*Complete* problem.

Based on these definitions, we can easily deduce that $\mathcal{P} \subseteq \mathcal{NP}$. However, the millennium prize problem whether $\mathcal{P} = \mathcal{NP}$ or $\mathcal{P} \subset \mathcal{NP}$ is still an open question. If $\mathcal{P} = \mathcal{NP}$ then $\mathcal{NP}$-*Complete* is empty, in other words, there exist polynomial algorithms to solve $\mathcal{NP}$ problems. This possibility is hard to accept, and hence, researchers consider always the hypothesis $\mathcal{P} \neq \mathcal{NP}$.

1.2.4 Graph classes

A graph structure is a scheme used to model different relationships between elements of the problem. Each element of the problem is associated with a mathematical abstraction called *vertices*, and each related pairs of vertices is called an *edge* (or an arc). Graphs represent a powerful tools to model and solve Combinatorial Optimization Problems.

Definition 1.2.8. In graph theory, a graph G is defined by a couple (V, E) where V is the set of vertices and $E \subseteq V \times V$ the set of edges connecting between vertices. The graph is said to be complete if $E = V \times V$. The neighborhood of a node i is defined as $N(i) = \{j | (i, j) \in E\}$ and the degree of node i is $|N(i)|$.

A graph is called non-oriented if the edges do not specify any direction. Otherwise, it is called directed graph.

Definition 1.2.9. In a directed graph G, a path is an ordered list of vertices $(v_1, v_2, \ldots, v_k)$ such that $(v_i, v_{i+1}) \in E \quad \forall i < k$. Moreover, If $v_1 = v_k$, the path is called a directed cycle.

Definition 1.2.10. Given a graph $G = (V, E)$, $G' = (V', E')$ is a sub-graph of G if it is obtained after removing some vertices and/or some edges from G, i.e. $V' \subseteq V$ and $E' \subseteq E$.

Definition 1.2.11. Let $G = (V, E)$ be a graph and let S be a subset of vertices in V. The induced graph $G[S]$ is a graph whose the vertex set is S and whose edge set consists of edges of E that have both endpoints in S.

Definition 1.2.12. Given a graph $G = (V, E)$, a *clique* of G is a subset of vertices $S \subseteq V$ such that $G[S]$ is a complete graph.

Definition 1.2.13. Given a graph $G = (V, E)$, an *independent set* of G is a subset of nodes $I \subseteq V$ such that $G[I]$ does not contain any edges.

Definition 1.2.14. Given a graph $G = (V, E)$, a *vertex-cover* of G is a subset of nodes $S \subseteq V$ such that the set $E_s = \{(i, j) \in E | i \in S \text{ or } j \in S\}$ is equal to E.

1.3 Resolution methods

Since solving $\mathcal{NP}$-*Hard* problems using polynomial algorithms seems to be impossible, efforts have been oriented toward the design of intelligent and effective methods. We provide in this section an overview on some of the different methods proposed during the past decades. Two main approaches exist to solve Combinatorial Optimization Problems : exact and approximate. Exact methods aim at finding the optimal solution, but when solving large-scale instances, they need much higher computational times. To tackle this problem, approximate methods are used. Although they do not guarantee the optimal solution, well-designed approximate methods can provide near optimal solutions in a polynomial time. Before introducing some methods of the two approaches, let us start first with preprocessing.

Preprocessing is a preliminary phase carried out on instances of a given $\mathcal{COP}$ in order to reduce the search space and probably ease the solution process. For example, by analyzing the nature of the objective function and the constraints along with the input data, the values of some variables X_i can be fixed or the size of D_i can be reduced. It is noteworthy to mention that the optimal solution after performing the preprocessing phase should also be the optimal solution for the original one.

1.3.1 Exact methods

Exact methods are designed in the purpose of finding the optimal solution for the problem being solved. In case where finding optimal solutions could be difficult, exact methods can also be used to obtain *theoretical bounds*. We present in the following the definition of *theoretical bounds* and also a brief description of the most used exact methods for solving vehicle routing problems.

Theoretical bounds

Bounds of a $\mathcal{COP}$ define boundaries between which the value of the optimal solution should occur. The tighter the boundaries are, the better are the bounds. Thus, the *upper* and the *lower bound* are defined.

Definition 1.3.1. For a given $\mathcal{COP}$ $\Phi = (\Omega, f)$ with a maximization objective function, an *upper bound* UB on the objective function is a value such that $\forall s \in \Omega$, $f(s) \leq UB$. In the other hand, a *lower bound* on the optimal objective value s^* is a value LB such that $s^* \geq LB$.

Definition 1.3.2. For a given $\mathcal{COP}$ $\Phi = (\Omega, f)$ with a minimization objective function, a *lower bound* LB on the objective function is a value such that $\forall s \in \Omega$, $f(s) \geq LB$). In the other hand, an *upper bound* UB on the optimal objective value s^* is a value such that $s^* \leq UB$.

For a maximization problem, *upper bounds* (UB) are obtained by solving a relaxed version of the problem, whereas *lower bounds* (LB) can be obtained using approximate methods. In the case where $UB = LB$ for an instance of the problem, the bounds represent also the optimal objective value.

Branch and bound scheme

The *branch and bound* scheme, proposed by [Clausen, 1999], is based on the idea of enumerating the solutions of the search space of a given $\mathcal{COP}$ $\Phi = (\Omega, f)$. The key feature of this method is to divide Ω into sub-spaces $(S_1, \ldots, S_k)$ which correspond

respectively to sub-problems $\{\Phi_1, \ldots, \Phi_k\}$ (*branching*). Each sub-space is in its turn divided recursively into sub-spaces. Evaluating (*bounding*) all the sub-problems and taking the global best solution is equivalent to solving the original problem. The sub-spaces generated during the search process can be assimilated to a tree search or a decision tree where each node represents a sub-problem of its parent node and the original problem is represented by the root node. The evaluation of each node i is carried out by computing LB_i and UB_i of the corresponding sub-problem Φ_i. For a maximization problem for example, if the UB_i of a given sub-problem Φ_i is lower than the best upper bound found LB_{best}, the corresponding node can be *pruned*. Hence, this method is generally faster than the brute force enumeration.

The efficiency of the method is generally mesured to the following factors :

- Bounds : upper and lower bounds.
- Dominance rules : a set of constraints used to reduce the search space. Note that the optimal solution must be guaranteed.
- Branching rules : strategies used to generate sub-spaces for possible exploration.
- Selection strategies : methods used to choose the most promising sub-spaces to explore first.

Linear programming - LP

Linear Programming (LP) is a method used to model continuous optimization problems where the objective function and problem constraints are modeled using linear expressions. Several methods have been proposed to solve LP models, mainly the *ellipsoid method* [Khachiyan, 1979] which has a polynomial time complexity. The most used method is the simplex method. Although proved to have exponential worst case complexity, it is widely used thanks to its quick convergence. Most of the commercial software (CPLEX, GUROBI, XPress, SCIP, etc.) implements this method.

By considering integrety constraints, LP can also be used to model $\mathcal{COP}$s and it is called in this case Integer Linear Programming (ILP). Adding such constraints makes solving ILP more difficult, generally NP-Hard. Figure 1.1 points out the difference between ILP and its associated LP (variables have real values). Since the LP is a relaxation for the original ILP, the search space of the LP includes that of the ILP. Figure 1.1 shows the difference between the search space of the ILP and that of the LP. Among the methods proposed to solve ILP models was the *cutting plane*

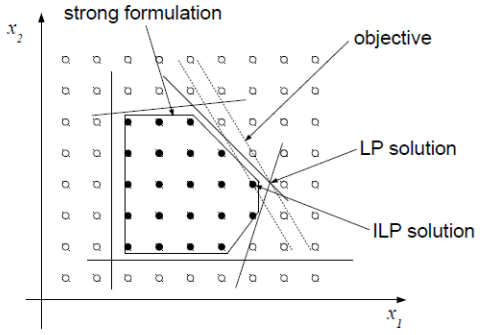


Figure 1.1 – Example of Mixed Integer Programming (MIP)

approach proposed by [Gomory et al., 1958]. The basic idea consists in solving ILP using simplex method. If the obtained solution is not in integers, a new constraint is systematically generated using the method called Gomory cuts. The new constraint is still satisfied by the optimal integer solution but not by the current non-integer solution. Once the additional constraint is added to the model, the current solution becomes non-feasible and the new model is solved again using simplex. This process is repeated until an integer solution is found.

Branch and cut algorithm - B&C

Although the proof of convergence of the *cutting plane* algorithm is given, the method has shown its limits in many cases. To overcome this problem, a combination of the *cutting plane* and the *branch and bound* methods was proposed [Mitchell, 2002] : cutting plane method is used to bring the relaxation closer to the integer programming problem, and the *branch and bound* uses a sophisticated divide and conquer strategy to solve the problem. Typically, the linear relaxation of the original ILP is solved using the simplex method. Subsequent relaxations in the search tree nodes are solved using the dual simplex method. In each node of the search tree, a set of separation routines are used to identify violated inequalities. If any of the inequalities is violated, it is added to the LP to cut off the non-feasible solution. Then the new LP is solved. The cuts can be dominance properties or valid inequalities. These cuts are satisfied by all the integer possible solutions. Branching occurs when no inequality is violated by the LP solution.

Column generation - CG

Some ILP formulations have an exponential number of variables. Solving such models using previous techniques leads to an exponential time complexity. Column Genera-

tion (CG) method proposed by [Dantzig and Wolfe, 1960] presents an efficient exact method for solving large linear problems. The main idea is to avoid the representation of all the variables (columns) of the whole problem, since at the optimum most of these variables will be non-basic. Only variables that potentially improve the objective function will be generated. The problem being solved is modeled by two problems : the master problem and the sub-problem. The master problem is the original problem with only a subset of variables being considered. The sub-problem, called the pricer, is the column generation procedure. It generates new variables that are likely to improve the current solution of the master problem. To achieve that, only variables with negative reduced costs are considered and added to the master problem. After adding the new variables to the master problem, the new ILP is solved. This process is repeated until no variables with negative reduced costs are found. The obtained solution is then optimal. The performance of the CG depends mainly on the technique used to generate columns at each iteration.

Branch and price - B&P

Branch and price method combines between the *branch and bound* technique with the column generation method [Barnhart et al., 1998]. The principle of the method is similar to that of the *branch and cut*, except that instead of generating rows in the nodes of the search tree (constraints), the focus is on column generation (variables). In each node the search tree, the linear relaxation of the problem is solved while considering only a subset of variables. Then, column generation technique is used to solve optimally the LP solution. First, the separation problem for the dual LP called the pricing problem is solved to check the optimality of the current solution. If new columns are identified to enter the basis, the LP is re-optimized. This process is repeated as long as columns price out to enter the basis. If the LP solution found at the end of the process does not satisfy the integrality constraints, the branching occurs.

Dynamic programming - DP

Dynamic Programming is a method used to solve a $\mathcal{COP}$ by breaking it down into a collection of sub-problems [Bellman, 1954]. In order to find the solution for the whole problem, we need first to solve the different parts of the problem and combine their solutions in a recursive fashion. A naive method would generate sub-problems many times, whereas in *Dynamic Programming*, each sub-problem is solved only once and stored in memory. To solve larger sub-problems, we need only to look up for smaller sub-problem solutions and combine them to get the new solution.

Constraint programming - CP

Constraint programming [Rossi et al., 2006] is a powerful programming paradigm for solving $\mathcal{COP}$s. CP relies on a wide range of techniques from operations research, computer science, artificial intelligence, graph theory, etc. CP was shown to be most effective on many combinatorial problems, such as timetabling, sequencing and resource-constrained scheduling. To model a $\mathcal{COP}$ using CP, a set of decision variables, each with a given domain of values, are defined. A set of constraints relating between the decision variables specify the properties of feasible solutions. Constraints in CP are of various forms : logical predicates, linear inequalities and others. Constraint solvers aim at finding an assignment to all the decision variables that satisfies the constraints in such a way that optimizes a given objective function. Constraint solvers incorporate generally several techniques, like backtracking, branch and bound algorithms, local search and constraint propagation. The latter is one of the most important concepts in CP paradigm [Rossi et al., 2006]. It consists in the propagation of the information contained in a constraint to the neighboring constraints. This techniques help to reduce the domain of values of decision variables and so the search space.

1.3.2 approximate methods

Advances in metaheuristics [Glover and Kochenberger, 2006] is one of the most outstanding achievements during the last decades in the field of the combinatorial optimization. Metaheuristics generally combine between several local search procedures and manage through higher level strategies in order to escape from local optima and perform a robust search of the solution space. We distinguish two main classes, metaheuristics that operates on one solution at time, called mono-solution, and those that operates on a set of solutions, called population-based metaheuristics. We provide in the following a rapid description of a sample of metaheuristics proven to be efficient to tackle $\mathcal{COP}$s in general and Vehicle Routing Problems in particular.

Greedy randomized adaptive search procedure - GRASP

First introduced by [Feo and Resende, 1995], it is basically a multistart process, where each iteration is made up from a randomized greedy construction heuristic to create a new solution, local search procedure is then applied on this solution to improve its quality. The best overall solution is then selected.

Iterated local search - ILS

ILS method [Lourenço et al., 2003] is based on the following idea : in each iteration, a solution is built using an embedded heuristic, but, instead of starting each time from scratch, the solution obtained in the previous iteration is used in the next one as an initial solution. This process can be seen as a single chain (sequence) of solutions being followed by the ILS. The search for a better solution is carried out in the neighborhood defined by the embedded heuristic. The latter is generally composed of a local search and a perturbation procedure.

Simulated annealing - SA

This heuristic [Kirkpatrick et al., 1983] is inspired from the physical annealing in metallurgy, a process that combines between heating and slow cooling of a material to find the wright temperature so that its atoms form crystals with no defects. By analogy, a solution s in combinatorial optimization corresponds to a state of the physical system and its objective value $f(s)$ to the energy of the system. In each iteration of the SA, the solution undergo a transformation (corresponding to a neighborhood operator). If the new solution s' improves the objective function, then it is accepted and becomes the current solution. Otherwise, the solution is accepted according to a probability called the Metropolis criterion $e^{\frac{f(s')-f(s)}{T}}$ where the parameter T represents the "temperature". The latter evolves during the search process by imitating the cooling process in metallurgy. When the temperature is high, it is more likely to accept the new solution even with a cost increase. Whereas when the temperature becomes low, only improving solutions are accepted.

Tabu search - TS

Tabu Search [Glover and Laguna, 1998] is a deterministic method. The main idea of TS is to enumerate at each iteration all the neighborhood of the incumbent solution and select the best one as the new solution, even if it increases the cost. In this way, TS can easily escape from local optima. To avoid cyclic behavior of such approach, a short term memory called tabu list is used to store recently visited solutions (or particular attributes of the recently visited solutions). The method stops after a certain number of iterations or after a maximum number of iterations without improvement to the best solution. TS performance can be improved by the introduction of intensification and diversification techniques. The implementation of those techniques is mainly based on long-term memories.

Variable neighborhood search - VNS

The key idea of VNS [Mladenović and Hansen, 1997] is the use of multiple neighborhoods instead of a single neighborhood as it is the case in local search heuristics. By having the ability to switch from a neighborhood to another, VNS can efficiently escape from local optima. The general scheme is as follows. A set of neighborhood structures $(N_1, \ldots, N_k)$, which are often nested, are defined and ordered according to their increasing size. An initial solution is randomly generated in the first neighborhood and improved by a local descent. If no improvement is found, the heuristic go to the next larger neighborhood. The local search returns to the first neighborhood when either the incumbent solution is improved or all the neighborhoods are explored. Upon termination, the returned solution is locally optimal with respect to all the neighborhoods. Many variant of VNS exist in the literature, we cite the reduced VNS, the skewed VNS and the Variable Neighborhood Descent (VND).

Adaptive large neighborhood search - ALNS

Adaptive large neighborhood search [Ropke and Pisinger, 2006] combines between several neighborhoods in a parallel fashion in order to explore the search space. Each neighborhood is defined implicitly by a *destroy* and a *repair* operator. The destroy operator removes part of the incumbent solution in a probabilistic way, and the repair operator is used to rebuild it. The set of solutions that can be generated by applying these two methods defines a neighborhood of solutions. In the ALNS, the choice of the destroy and the repair operator in each iteration is performed in a dynamic way based on their contribution to the search progress. Technically, a weight is assigned to each operator to control how often the operator is used during the search. The weights are modified and adjusted according to the effectiveness of each operator in order to allow the ALNS to adapt to the instance being solved and to the state of the search.

Genetic algorithm - GA

Genetic algorithm [Holland, 1992] is a population-based metaheuristic widely used in many fields of computer science. This method belongs to the class of evolutionary algorithms which simulate natural evolution by the use of operators that imitate those found in nature, namely, selection, mutation, inheritance and crossover. GA precisely is motivated by the Darwinian principle of natural selection in genetics. The basic idea is to evolve a population of individuals from one generation to the next through crossover and mutation phenomena. Technically, the initial population

is composed of individuals represented by "chromosomes", a bit or integer strings. In each iteration, a subset of chromosomes called "parents" are selected through a probabilistic selection process and used for reproduction. The reproduction mechanism, called crossover, combines between two parents to create one or two offspring chromosomes having the best traits of their parents. With a low probability, the new children will be modified using a mutation operator in order to diversify the population and avoid early convergence. Two main approaches are used to update the current population. The first one, called *generational genetic algorithm*, in which only offspring population are used to form the new population. In the second approach, called *steady-state genetic algorithm*, offspring chromosomes compete with the current population and the best ones, according to their fitness, are used to form the new population.

Memetic algorithm - MA

Despite the fact that the average quality of the population is improved over the generations, GAs fail to find near-optimal solutions. The main reason is that crossover and mutation operators are used mainly as diversification techniques and do not reinforce enough the intensification of the search procedure. Based on these observation, [Moscato et al., 2004] introduced an improved scheme of GA called the Memetic algorithm (MA). The MA consists of a hybrid between a global search, carried out by a classical metaheuristic, and the local search procedures. The two components are in fact complementary, the GA is used to detect the most promising regions in the search space, whereas the local search enhance the intensification process in these regions.

Particle swarm optimization - PSO

Particle swarm optimization [Eberhart and Kennedy, 1995] is one of the metaheuristics inspired from the social behavior of animals evolving in swarm. Its basic idea is to simulates the way animals find the food and how they elaborate collectively the orientation from the nest to food sources. An individual of the group is represented by a particle and the solutions represent the positions or the search areas to be explored by the particles. Each particle stores its current position as well as the best position found so far during the search. The latter represents the individual experience of the particle, whereas the best position found by the population represents the group experience. Each particle is characterized by a movement speed which indicates the degree of change to be made on its current position. This speed is updated in each iteration with respect to the following factors : 1) its current

speed weighted by a constant w, called the *inertia* factor, 2) the trend to return to its personal experience weighted by a constant c_1 (*cognitive factor*), 3) the trend to join the group experience weighted by a factor c_2 (*social factor*).

Ants colony optimization - ACO

This metaheuristic [Dorigo et al., 1996] simulates the behavior of ants while looking for the shortest path to the food sources. This process is based on lying down on ground a chemical compound called "pheromone" in order to communicate information about the good paths. Basically, a number of ants construct solutions using a randomized and greedy heuristic. The selection of the next element to be incorporated in the current solution is based mainly on a heuristic evaluation, but also the amount of pheromone related to that element. The pheromone models the memory of the system related to the presence of that element in previously constructed solutions. That is, elements with good heuristic evaluations and a high level of pheromone are more likely to be chosen.

1.4 Vehicle routing problems and its variants

Vehicle routing problem- VRP [Toth and Vigo, 2014] is one of the most studied problems in combinatorial optimization. The attention paid for VRPs is justified by the several applications mainly found in transportation and logistics. These fields are vital to modern economy, since they guarantee the flow of goods and services. Basically, in a VRP problem, a fleet of vehicles is available to visit a set of customers. A visit of a customer by a given vehicle is used to model the fulfillment of a service. The customers are generally situated on the plane, but also in the space. The objective in VRP and its variants is generally to schedule the visits of the customers in a way that optimizes the use of the available resources and with respect to some constraints.

In this section, we introduce the problems of vehicle routing in general and we give some necessary details used later in this manuscript. We discuss then some of its main variants that exist in the literature. After that, we will take a closer look into routing problems with profits and different related aspects treated in this thesis.

Traveling salesman problem - TSP The most classic and the simplest among routing problems is the Traveling Salesman Problem (TSP) [Laporte, 1992]. It describes the situation where a salesman must visit N cities and return back to its departure city while minimizing the total travel distance of the journey. TSP has

been proven to be NP-Hard using a reduction to the Hamiltonian Cycle Problem HCP [Laporte, 1992]. Formally, TSP can be modeled using a complete directed graph $G = (V, A)$ where $V = \{1, \ldots, n\}$ are the vertices representing the set of cities and $A = \{(i, j) \in V^2\}$ is the set of arcs which represents the routes relating between each couple of cities. A cost c_{ij} is associated with each arc to model the distance or the travel time needed to traverse it. The objective in TSP is to find the shortest cycle that visits each city exactly once.

Vehicle routing problem - VRP The vehicle routing problem is a generalization of the TSP where m identical vehicles with a limited capacity Q are used to visit the customers. Each vehicle starts from a node-depot, visits a subset of customers with respect to the capacity limitation, and returns to the depot. The objective is to satisfy the demand of all the customers while minimizing the overall transport costs of the vehicles. Many exact methods have been proposed for VRP, mainly based on branch-and-cut and column generation [Toth and Vigo, 2014]. Regarding approximate methods, TS [Gendreau et al., 1994] [Taillard, 1993], SA [Osman, 1993] and ACO [Bullnheimer et al., 1999] [Kawamura et al., 1998] [Yu et al., 2009] were proposed.

Vehicle routing problem with time windows - VRPTW The vehicle routing problem with time windows is the most studied variant of VRP [Toth and Vigo, 2014]. In this variant, the service of each customer must start within a given time interval, called a *time window*. A time window is defined for each customer i by an earliest service time o_i and a latest service time c_i. Moreover, a service time s_i is commonly associated with the visit of each customer i. If a vehicle arrives at customer i before o_i, it results in a waiting time. On the other hand, a vehicle is not allowed to visit a customer after the specified latest time c_i. Some variants of VRPTW consider *soft time windows*. In this case, a time window is only a preference which may be violated at the expense of penalty costs. Many exact and heuristic methods have been proposed for the VRPTW. The main exact methods proposed are : branch and cut [Bard et al., 2002], branch and price [Feillet et al., 2007] and branch and cut and price [Jepsen et al., 2008]. Regarding metaheuristics, we can find TS [Cordeau et al., 2001], SA [Czech and Czarnas, 2002] and GA [Berger and Barkaoui, 2004].

In this thesis, we focused on vehicle routing problem with profits. This class of problems are described in the next section.

Vehicle routing problems with profits - VRPPs

VRP with profits (VRPPs) is a variant in which it is usually impossible to serve all the customers. To discriminate between customers, the visit of each customer is rewarded with a profit. Profits represent in this context the value gained by serving the customers. In addition, like VRPs in general, tours should be of the minimum cost. Hence, VRPPs can be seen as a bi-objective problems where the collected profit is to maximize and the the cost of tours to be minimized. According to this definition, many variants can figure in this class of problems. Three main categories are identified [Archetti et al., 2014] : the *Orienteering Problem* - OP, where the objective is to maximize the collected profit with respect to a travel distance limitation. The *Prize Collecting TSP* - PCTSP, which aims to minimize the travel cost while guaranteeing a minimum collected profit. The *Profitable Tour Problem* - PTP where the goal is to maximize the difference between collected profit and costs.

Among these three categories, the OP, known also as the Selective Traveling Salesman Problem, is one of the most studied problems in the literature [Angelelli et al., 2014]. This problem is inspired from the sport game of orienteering as described in [Chao et al., 1996]. Many exact and heuristic methods have been proposed for the problem. The reader can refer to [Feillet et al., 2005],[Archetti et al., 2014] and [Vansteenwegen et al., 2011] for excellent surveys on these variants, applications and also resolution methods.

The generalization of OP to multiple vehicles gives rise to the *Team Orienteering Problem* - TOP [Chao et al., 1996]. This problem was first proposed in [Butt and Cavalier, 1994] under the name of *Multiple Tour Maximum Collection Problem*. In this problem, a set of customers N need to be served, and a profit is associated with each customer. A fleet of m identical vehicles with a limited travel time T_{max} is available to serve the set of customers. Each customer can be visited at most one time, and its profit is gained in return. The objective then of the TOP is to find a subset of customers to visit by the vehicles so that total collected profit is maximized. The main exact methods proposed for solving the TOP are CG [Butt and Ryan, 1999], B&P [Boussier et al., 2007], branch and cut and price [Poggi et al., 2010] and cutting planes [El-Hajj et al., 2016]. Several metaheuristics have been also proposed for the TOP, we cite MA [Bouly et al., 2010], PSO [Dang et al., 2013], VNS/TS [Archetti et al., 2007], ACO [Ke et al., 2008] and PR [Souffriau et al., 2010].

We give in the following a brief description of the variants treated in this manuscript. The first one is called the *Clustered Team Orienteering Problem*. The

second variant is the *Set Orienteering Problem*. The last variant is called the *Synchronized Team Orienteering Problem with Time Windows*.

Clustered Team Orienteering Problem - CluTOP

The *Clustered Team Orienteering Problem* (CluTOP) is a generalization of the TOP, and in the same time a generalization to multiple vehicles for the *Clustered Orienteering Problem* (COP), recently introduced in [Angelelli et al., 2014]. The COP is a variant of the OP in which, customers are grouped into subsets called *clusters*. Each cluster is characterized by a profit which is gained only if all of its customers are served. The single available vehicle in the COP has a limited travel time T_{max} in order to visit the customers. The objective is to visit the customers of a subset of clusters in order to maximize the total collected profit while respecting the time limit constraint. Note that there is no restriction on the order of visits, i.e. the vehicle can visit customers from cluster a, visits customers from cluster b and then returns and continues to visit other customers from cluster a. In the case of multiple vehicles (CluTOP), a fleet of m of identical vehicles is used to serve the clusters. To that end, the vehicles collaborate in order to visit the customers of the chosen clusters. One vehicle can visit customers from several clusters, in arbitrary order, and the customers of a given cluster can be visited by using more than one vehicle.

Set Orienteering Problem

The *Set Orienteering Problem* (SOP) is a new variant of the OP recently introduced by [Archetti et al., 2018]. Customers are grouped into subsets called *clusters*. Each cluster is assigned a profit which is gained only if at least one of its customers is served. A single vehicle is available to serve the customers. The vehicle starts the journey from the depot, visits the customers and returns back to the depot while respecting a limited travel time. Due to this constraint, it is impossible to serve all the clusters. The objective then is to visit as much customers as possible in order to maximize the total collected profit. the SOP can be seen also as the selective version of the Generalized Traveling Salesman Problem (GTSP). The SOP finds its applications in mass distribution products where a carrier should serve a set of retailers belonging to different supply chains. In order to reduce distribution costs, a chain can stipulate in the contract to receive the required quantity of products all at once in one of its retailers instead of visiting all of them. The inner distribution is let to the chain. Another application is when the customers are clustered into sub-areas, and the service of one sub-area is achieved by visiting only one of its

customers.

Synchronized Team Orienteering Problem with Time Windows

The *Synchronized Team Orienteering Problem with Time Windows* (STOPTW) is a recent variant of TOP with additional constraints, namely the time windows and the synchronized visits. The STOPTW was originally introduced by [Roozbeh et al., 2018] in order to model the asset protection problem during escaped wildfires. In this problem, a fleet of heterogeneous vehicles is available to serve a given set of customers. Each type of vehicles has a limited number of vehicles and its own transportation network. A customer is characterized by a profit, a service duration and a time window. Each customer requires to be served by a certain number of vehicle per type. In order to gain the profit of a given customer, it must be simultaneously visited by the required number of vehicles. The objective is to serve as much customers as possible while respecting time windows and synchronization constraints. The STOPTW was originally proposed as a tool to model the situation where a set of assets need to be protected during escaped wildfires. Each customer represents an asset. The profit of each customer is used to model the importance of its associated asset. Time windows are used to simulate the propagation of fire fronts across the landscape. In order to protect a given asset, a specific equipment and firefighting trucks are needed to perform protection activities simultaneously within the corresponding time window. The objective is to protect as much assets as possible in a such a way that the total value of saved assets is maximized under synchronization and time window constraints.

1.5 Conclusion

We recalled in this first chapter some fundamental and necessary concepts in combinatorial optimization research field. We started by giving a formal definition for Combinatorial Optimization Problem. Then, we presented algorithms complexity and the $\mathcal{NP}$-Completeness theory. We also listed the most used resolution methods to solve $\mathcal{COP}$s. Finally, we gave a brief description of Vehicle Routing Problems, and focused on Team Orienteering Problem. In the next chapters, we present our contribution and resolution methods proposed for the TOP variants tackled in this thesis.

The Clustered Team Orienteering Problem

Sommaire

2.1 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 problem, 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 [Archetti et al., 2014]. 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 [Chao et al., 1996]. Many exact and heuristic methods have been proposed for the problem. The reader can refer to [Archetti et al., 2014], [Feillet et al., 2005],

[Gunawan et al., 2016] and [Vansteenwegen et al., 2011] 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. [Angelelli et al., 2014]. In this problem, a single vehicle is used to serve the selected clusters. The authors in [Angelelli et al., 2014] 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) [Fischetti et al., 1997], 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 [Jongens and Volgenant, 1985], customers belonging to the same *cluster* must be visited contiguously. An extension of these two problems has been introduced in [Defryn and Sörensen, 2017], 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 [Archetti et al., 2018]. In this problem, a single vehicle can visit one customer per cluster at most. To the best of our knowledge, no paper in the literature has ever studied the CluTOP as defined in this chapter.

In real life, the CluTOP can be used to model many applications in logistic systems and transportation. Some interesting applications were introduced in [Angelelli et al., 2014] 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.

The contributions in this chapter are two-fold :

- 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.

- In terms of computational experiments, tests were conducted on benchmarks proposed by [Angelelli et al., 2014] to show the efficiency of our methods. Eighty additional instances were solved to optimality by the cutting plane. Furthermore, we present the results of extensive computational experiments on new proposed CluTOP instances where $m \geq 2$.

2.2 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 time denoted, as appropriate, c_e or $c_{(i,j)}$. These travel times 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 customers shared 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 fleet of m identical vehicles with a limited travel time T_{max} is available to serve the customers. It is worth mentioning that any vehicle can visit customers from different clusters, and the customers of a given cluster can be visited by different 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 2.1 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 :

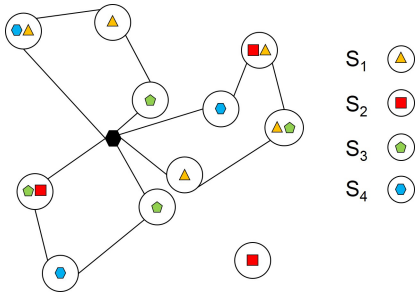


Figure 2.1 – Example of a CluTOP solution

- z_k : equal to 1 if all customers in cluster $S_k \in S$ are visited, 0 otherwise.
- y_{ir} : equal to 1 if vertex $i \in V$ is visited 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 :

$$max \sum_{S_k \in S} P_k z_k \tag{2.1}$$

$$\sum_{r=1}^{m} y_{ir} \leq 1 \quad \forall i \in V \backslash \{0\} \tag{2.2}$$

$$\sum_{e \in \delta(i)} x_{er} = 2y_{ir} \quad \forall i \in V, r \in \{1, \ldots, m\} \tag{2.3}$$

$$\sum_{e \in E} c_e x_{er} \leq T_{max} \quad \forall r \in \{1, \ldots, m\} \tag{2.4}$$

$$\sum_{e \in E(U)} \sum_{r=1}^{m} x_{er} \leq \sum_{i \in U \backslash \{t\}} \sum_{r=1}^{m} y_{ir} \quad \forall U \subseteq V \backslash \{0\}, \quad \forall t \in U \tag{2.5}$$

$$z_k \leq \sum_{r=1}^{m} y_{ir} \quad \forall S_k \in S, \quad \forall i \in S_k \tag{2.6}$$

$$z_k \in \{0, 1\} \quad \forall S_k \in S \tag{2.7}$$

$$x_{er} \in \{0, 1\} \quad \forall e \in E, \quad \forall r \in \{1, \ldots, m\} \tag{2.8}$$

$$y_{ir} \in \{0, 1\} \quad \forall i \in V, \quad \forall r \in \{1, \ldots, m\} \tag{2.9}$$

The objective function (2.1) maximizes the total collected profit. Constraints (2.2) 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 (2.3) are the flow conservation constraints. Constraints (2.4) guarantee that the travel time of each vehicle does not exceed T_{max}. Constraints (2.5) are subtour elimination constraints (SECs). Constraints (2.6) together with the objective function state that the profit of a cluster is gained only if all of its customers are served. Constraints(2.7)-(2.9) are domain definition.

2.3 Valid inequalities

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.

$$\sum_{r=1}^{m} y_{ir} \leq \sum_{S_k \in \zeta(i)} z_k \quad \forall i \in V \backslash \{0\} \tag{2.10}$$

This constraint can be considered as part of the basic model. In fact, when triangle inequality is verified, any additional customer in the optimal solution cannot improve the total travel time, and hence, it does not improve the objective value.

2.3.1 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 2.3.1. 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 different incompatibilities that can exist between each pair of customers, we define 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 2.3.1. *Let $\mathcal{C}$ denote the set of maximal cliques of the incompatibility graph G^{inc}. The following inequalities :*

$$\sum_{i \in C} y_{ir} \leq 1 \quad \forall C \in \mathcal{C}, \forall r \in \{1, \ldots, m\} \tag{2.11}$$

are valid for $ILP0$

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

Definition 2.3.2. Two clusters S_k and S_l are said incompatible if and only if it is impossible to serve all their customers 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 using the m available vehicles. Hence the following proposition holds :

Proposition 2.3.2. *Let $\mathcal{C}_{cl}$ be the set of maximal cliques of the incompatibility graph G^{inc}_{cl}. The following inequalities are valid for $ILP0$.*

$$\sum_{S_k \in C} z_k \leq 1 \quad C \in \mathcal{C}_{cl} \tag{2.12}$$

2.3.2 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 between 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 $p_i = \sum_{i \in S_k} \rho_{ik}$ 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 :

$$\sum_{i \in V \setminus \{0\}} y_{i(r+1)} p_i - \sum_{i \in V \setminus \{0\}} y_{ir} p_i \leq 0 \quad \forall r = \{1, \ldots, m-1\} \tag{2.13}$$

2.3.3 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 2.3.3. 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 :

$$\sum_{k=1}^{K} P_k z_k \leq \sum_{l:S_l \in \phi^*} P_l \tag{2.14}$$

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 P_r as total profit. Therefore, the following constraint holds :

$$\sum_{S_k \in S} z_k \geq LB_{cl} \tag{2.15}$$

2.4 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.

2.4.1 Global scheme

We start by describing the global scheme of our algorithm. In order to solve the CluTOP, we are interested in solving $ILP0$. Moreover, we include the valid inequalities (2.10)-(2.15). 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 (SECs) 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 are 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 SECs are generated and added to the model. In our work, we use a specific type of SECs called Generalized Subtour Elimination Constraints (GSECs) proposed in [Fischetti et al., 1997]. The GSECs are defined as follows :

$$\sum_{e \in \delta(U)} x_{er} \geq 2y_{ir}, \quad \forall U \subset V, 0 \in U, \forall i \in V \backslash U, r \in \{1, \ldots, m\} \tag{2.16}$$

$$\sum_{e \in E(U)} x_{er} \leq \sum_{i \in U} y_{ir} - y_{jr}, \quad \forall U \subset V, 0 \in U, \forall j \in V \backslash U, r \in \{1, \ldots, m\} \tag{2.17}$$

$$\sum_{e \in E(U)} x_{er} \leq \sum_{i \in U} y_{ir} - y_{jr}, \quad \forall U \subseteq V \backslash \{0\}, \forall j \in U, r \in \{1, \ldots, m\} \tag{2.18}$$

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 [El-Hajj et al., 2016].

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
Execute the pre-processing procedure(see Section 2.4.3)
Add the computed valid inequalities and relax constraints (2.5) 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 2.4.2)
until (S*_ILP1 is optimal for CluTOP or time expired)
```

2.4.2 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 subtours that 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 those with a small number of customers located in the 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.

2.4.3 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 (provided later Section 3.1).

2.4.3.1 Inaccessible components

Definition 2.4.1. 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 2.4.2. 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.

$$y_{ir} = 0 \quad \forall i \in V \backslash \{0\}, \text{ and } c_{(0,i)} + c_{(i,0)} > T_{max}, \forall r = 1 \dots m \tag{2.19}$$

$$z_k = 0 \quad \forall k = 1, \dots, K \text{ and } S_k \text{ is inaccessible} \tag{2.20}$$

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.

2.4.3.2 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.

2.4.3.3 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 metaheuristic proposed in [Dang and Moukrim, 2012].

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 2.4.1. *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 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 2.4.2. *If $UB(I_{kl}) < P_k + P_l$ then S_k and S_l are incompatible.*

Finally, we describe the lower bound on the number of clusters LB_{cl} that should be served in the optimal solution. From a $CluTOP$ instance I, we derive an instance I_{cl} in which each cluster has a profit 1. To solve this instance, we use the same approach used to solve $ILP0$. Indeed, we consider a modified version of $ILP0$,

where the constraint (2.22) is added to the model and the objective function is (2.21) instead of (2.1).

$$min \sum_{S_k \in S} z_k \tag{2.21}$$

$$\sum_{S_k \in S} P_k z_k \geq LB(I) \tag{2.22}$$

Algorithm 2 details the pre-processing phase.

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 2.3.3 and 2.3.3)
Calculate inaccessible customers and clusters (See Section 2.4.3.1)
Calculate mandatory clusters (See Section 2.4.3.2)

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

2.5 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 algorithm on two different problem sets. The first set (Set A) concerns the instances introduced in [Angelelli et al., 2014] for the case of a single vehicle. The second set (Set B) is related to the multiple vehicles. In the following, we provide a detailed description of these sets and we report the results of our computational experiments.

2.5.1 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 [Angelelli et al., 2014].

To do this, we use benchmark instances introduced in [Angelelli et al., 2014].

2.5.1.1 Description of the instances

The benchmark is derived from 50 instances of TSPLIB with a number of vertices ranging from 42 to 318. 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.

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 [Fischetti et al., 1998]. 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 : $\frac{1}{2}$ and $\frac{3}{4}$.

As a result, 16 different instances are derived from each TSP benchmark instance. As a result, 800 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. [Angelelli et al., 2014].

2.5.1.2 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. [Angelelli et al., 2014]. In Table 2.1, 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 the exact method respectively.
- CPU : average CPU time in seconds.

From Table 2.1, 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.

Tableau 2.1 – Performance of the cutting plane

Class	COP-BASIC			COP-CUT			Cutting plane		
	#OPT	*OptGap*	*CPU*	*#OPT*	*OptGap*	*CPU*	*#OPT*	*OptGap*	*CPU*
*dantzig*42	16	0	8.13	16	0	0.61	16	0	2.90
*swiss*42	16	0	6.58	16	0	0.95	16	0	4.08
*att*48	16	0	13.18	16	0	8.45	16	0	34.80
*gr*48	16	0	11.88	16	0	4.86	16	0	12.46
*hk*48	16	0	10.87	16	0	5.54	16	0	12.42
*eil*51	16	0	16.71	16	0	6.76	16	0	16.56
*berlin*52	16	0	12.02	16	0	2.30	16	0	9.96
*brazil*58	16	0	16.44	16	0	5.47	16	0	26.67
*st*70	16	0	52.68	16	0	57.70	16	0	95.71
*eil*76	16	0	20.16	16	0	18.17	16	0	25.83
*pr*76	16	0	94.41	15	0.005	501.65	16	0	196.43
*gr*96	16	0	34.80	16	0	23.26	16	0	37.23
*rat*99	16	0	232.35	16	0	93.09	16	0	159.30
*kroA*100	14	0.037	1112.48	16	0	279.00	16	0	240.96
*kroB*100	11	0.070	1386.00	16	0	468.64	16	0	272.45
*kroC*100	11	0.072	1967.99	16	0	602.79	15	0.005	746.34
*kroD*100	14	0.021	1613.80	16	0	647.23	16	0	581.61
*kroE*100	13	0.028	1107.84	15	0.021	562.51	16	0	179.06
*rd*100	10	0.038	1702.83	16	0	537.00	16	0	225.11
*eil*101	16	0	93.98	16	0	50.94	16	0	115.96
*lin*105	16	0	104.36	16	0	59.40	16	0	72.66
*pr*107	16	0	171.29	16	0	61.62	15	0.019	332.52
*gr*120	10	0.048	1854.87	15	0.018	948.18	15	0.006	688.78
*pr*124	16	0	139.67	16	0	78.51	16	0	100.51
*bier*127	16	0	283.76	16	0	97.43	16	0	92.16
*ch*130	5	0.117	2717.58	12	0.026	1844.03	15	0.0042	767.34
*pr*136	11	0.046	2024.14	16	0	765.13	16	0	326.29
*gr*137	16	0	144.86	16	0	62.89	16	0	149.90
*pr*144	16	0	229.02	16	0	119.39	16	0	135.73
*ch*150	1	0.372	3438.73	0	0.444	3600.71	9	0.0584	2098.01
*kroA*150	2	0.325	3488.52	3	0.328	3232.80	10	0.0583	1713.36
*kroB*150	1	0.333	3591.24	1	0.346	3571.16	11	0.0563	1913.78
*pr*152	16	0	266.37	16	0	129.95	16	0	559.19
*u*159	13	0.013	923.46	16	0	442.11	14	0.0109	1185.79
*si*175	16	0	844.04	16	0	424.51	16	0	315.50
*brg*180	16	0	597.49	16	0	141.47	16	0	155.60
*rat*195	5	0.075	3020.09	7	0.081	3003.96	10	0.0447	1757.66
*d*198	16	0	179.49	14	0.014	1035.47	12	0.0332	1431.59
*kroA*200	0	0.625	3600.76	0	0.746	3600.97	6	0.1473	2562.21
*kroB*200	2	0.456	3471.47	0	0.642	3600.41	6	0.1767	2550.03
*gr*202	13	0.013	814.35	13	0.014	1255.61	16	0	557.13
*ts*225	0	0.369	3600.52	0	0.712	3600.40	0	0.2729	3600.73
*tsp*225	1	0.262	3504.62	3	0.148	3335.69	10	0.0681	2461.35
*pr*226	10	0.110	1782.04	14	0.019	1485.83	5	0.1441	3066.90
*gr*229	16	0	299.62	13	0.021	1645.25	16	0	1063.57
*gil*262	0	0.614	3601.48	0	0.918	3601.64	4	0.1986	3084.40
*pr*264	8	0.066	2499.76	10	0.175	2105.14	2	0.3496	3155.57
*a*280	1	0.176	3415.74	1	0.235	3464.00	2	0.1768	3357.49
*pr*299	1	0.176	3569.61	2	0.548	3554.80	1	0.2837	3449.92
*lin*318	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

In order to bring a more detailed picture of the performance of the cutting plane algorithm, we present in Table 2.2, a pairwise comparison between the considered methods. In this Table, we provide for each pair of exact method (EM) EM_{row} and EM_{col} that are displayed in some given row and column, respectively, the number of instances solved by EM_{row} but not by EM_{col}. From Table 2.2 we see that our method succeed to solve 75 instances against 34 solved by the COP-BASIC. Whereas for the cutting plane solves 107 instances against 22 solved by COP-CUT.

Tableau 2.2 – Pairwise comparison between exact methods

	COP-BASIC	COP-CUT	Cutting plane
COP-BASIC	-	60	34
COP-CUT	16	-	22
Cutting plane	75	107	

The authers in [Angelelli et al., 2014] reported a comparison study between COP-BASIC and COP-CUT while considering only 192 instances with a number of customers ranging from 200 to 318. They presented a detailed analysis of the COP-CUT and COP-BASIC without considering instances with less than 200 vertices. In this section, we proceed similarly in order to have a better picture of the performance of each method and to study their behavior according to the instances characteristics.

We start this study by giving, in Table 2.3, a general picture of the three methods on the subset of considered instances. Clearly, the results depicted in 2.3 strongly support the first results. Indeed, our method is still performing better than the branch-and-cut algorithms in terms of optimality gap and the total number of solved instances while requiring less CPU-time.

Tableau 2.3 – Results for large-scale instances

	COP-BASIC	COP-CUT	Cutting plane
Optimality Gap	0.393	0.251	0.164
#Opt	56	53	69
Average CPU	2904.19	2808.66	2695

Figures 2.2 to 2.7 show a comparison in terms of optimality gap and computational time with respect to different criteria : the number of clusters, the profit

generation pattern and the value of θ. In figure 2.3, g1 stands for the first type of profit generation pattern, whereas g2 are instances with the second type of profits. In figure 2.4, $q2$ and $q3$ stand respectively for values $\frac{1}{2}$ and $\frac{3}{4}$ of θ.

From Figures 2.2, 2.3 and 2.5, we remark that the performance of methods vary according to the different studied characteristics. Interestingly, the performance of our method does not seem to reflect these characteristics. For example, the percentage gap of cutting plane is quietly the same (between 4% and 7%) while the number of clusters varies from 10 to 25 (figure 2.2).

However, from figures 2.5, 2.6 and 2.7, we observe a slight disparity in computational time performance of our method when studying the impact of the different parameter instances on the CPU time.

- The profit generation has less impact on computational time in the case of COP-BASIC and COP-CUT as shown in figure 2.6, whereas computational time of the cutting plane increases for the second type of profit generation. However, our method remains competitive with the other methods. We note that in these instances, the profits are more heterogeneous than those of the first pattern. This may explain the fact that our method needs more computational time to solve these instances.

- Regarding the values of θ, we note from figure 2.7 that computational time of COP-BASIC and COP-CUT decrease as the value of θ increases from $\frac{1}{2}$ to $\frac{3}{4}$. The authors in [Angelelli et al., 2014] suggested that when $\theta = \frac{1}{2}$, the travel time constraint is more binding. As a result, the number of clusters to serve is smaller and the number of clusters to discard from the solution becomes bigger. According to the authors, this situation makes decisions about clusters to serve or not more complex than the case with $\theta = \frac{3}{4}$. On the contrary, computational time of our method increases as the value of θ increases. This is due to the fact when the T_{max} is bigger, the more the routing problem becomes hard to the cutting plane. In selective routing problems, there are two main sub-problems, the selection problem and the routing problem. Based on the observations above, we can deduce that COP-BASIC and COP-CUT are more sensitive to the complexity of the selection problem, whereas, the cutting plane is more sensitive to the complexity of the routing problem.

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

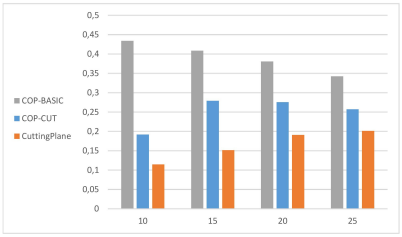


Figure 2.2 – Optimality Gap with respect to the number of clusters

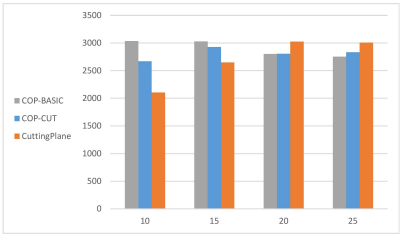


Figure 2.5 – Computational time with respect to the number of clusters

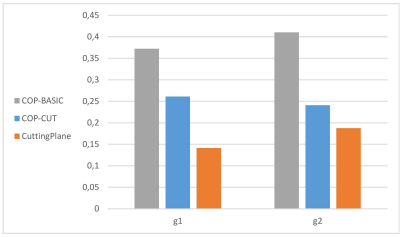


Figure 2.3 – Optimality Gap with respect to the profit generation

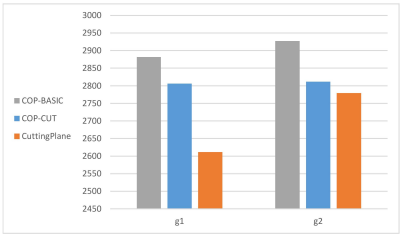


Figure 2.6 – Computational time with respect to profit generation

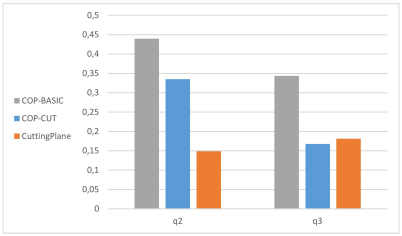


Figure 2.4 – Optimality Gap with respect to θ

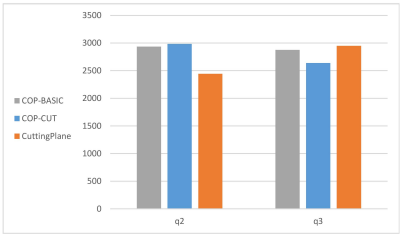


Figure 2.7 – Computational time with respect to to θ

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 substructures like meshes or even a set of co-linear points. Pferschy et al. pointed out in [Pferschy and Staněk, 2017] 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. [Angelelli et al., 2014] 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 2.4, 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. $LBGAP$).

On the basis of Table 2.4, we observe that :

- The poor performance of the exact methods presented in Angelleli et al. [Angelelli et al., 2014] is explained not only by the quality of the feasible solution (as suggested by [Angelelli et al., 2014]) 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 are due to the poor performance of both the lower and upper bound compared to Angelleli et al. [Angelelli et al., 2014].

Tableau 2.4 – 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

2.5.2 Set B : multiple vehicles problem set

2.5.2.1 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 [Chao et al., 1996]. 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 [Angelelli et al., 2014], we derived two instances with a number of vehicles two or three, knowing that T_{max} values in [Angelelli et al., 2014] 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 800 for each number of vehicles.

2.5.2.2 Performance of the exact method

Like in Section 2.5.1, 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 2.5 shows the performance of the cutting plane when considering one, two and three vehicles. From Table 2.5, 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. 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.

Tableau 2.5 – 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

continued on next page

Class	Single vehicle			Two Vehicles			3 Vehicles		
	#OPT	*OptGap*	*CPU*	*#OPT*	*OptGap*	*CPU*	*#OPT*	*OptGap*	*CPU*
*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
*d*198	12	0.0332	1431.59	8	0.191	1801.94	16	0	2.27
*kroA*200	6	0.1473	2562.21	2	0.594	3171.66	8	0.583	1802.31
*kroB*200	6	0.1767	2550.03	2	0.622	3164.14	8	0.601	1802.45
*gr*202	16	0	557.13	0	0.397	3600.87	2	0.351	3436.75
*ts*225	0	0.2729	3600.73	0	0.742	3601.16	8	0.496	1908.84
*tsp*225	10	0.0681	2461.35	0	0.499	3600.76	0	0.607	3600.47
*pr*226	5	0.1441	3066.90	8	0.179	2014.41	16	0	210.16
*gr*229	16	0	1063.57	0	0.605	3601.41	0	0.686	3602.57
*gil*262	4	0.1986	3084.40	0	0.827	3600.66	8	0.711	1801.03
*pr*264	2	0.3496	3155.57	8	0.281	1817.96	10	0.049	1358.66
*a*280	2	0.1768	3357.49	0	0.461	3600.39	0	0.518	3600.16
*pr*299	1	0.2837	3449.92	0	0.509	3600.8	2	0.402	3151.43
*lin*318	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

We investigate now the impact of the use of multiple vehicles on the performance of the cutting plane. Figures 2.8 to 2.13 depict some performance measurements 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.

2.6 Conclusion and future work

In this chapter, 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 generalizes also the Clustered Orienteering Problem (COP) where only a single vehicle is used. In order to solve the CluTOP, we proposed an exact method based on cutting planes approach. The results for in the case of a single vehicle show the competitiveness of our method compared to the literature, by achieving minimal gaps for the majority of classes along with new instances solved to optimality.

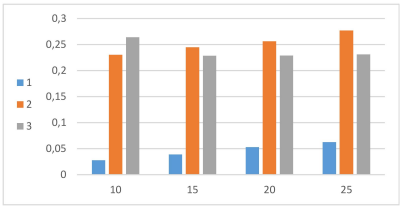


Figure 2.8 – Optimality Gap with respect to the number of clusters and the number of vehicles.

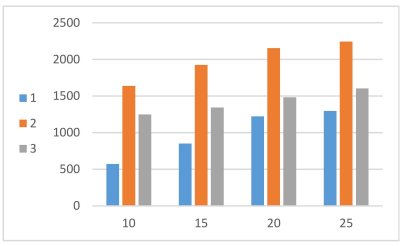


Figure 2.11 – Computational time with respect to the number of clusters and the number of vehicles.

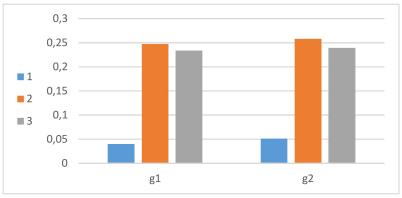


Figure 2.9 – Optimality Gap with respect to the number of profit generation and the number of vehicles.

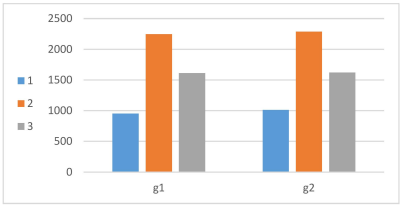


Figure 2.12 – Computational time with respect to profit generation and the number of vehicles.

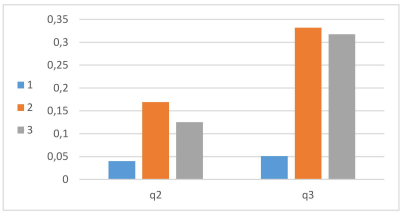


Figure 2.10 – Optimality Gap with respect to θ and the number of vehicles.

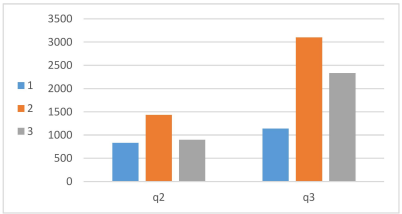


Figure 2.13 – Computational time with respect to θ and the number of vehicles.

Hybrid method for the Clustered Team Orienteering Problem

Sommaire

In this chapter, we propose a hybrid heuristic scheme based on the *order first-cluster second* approach [Prins et al., 2014] to solve the CluTOP. The first component is a metaheuristic 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 chapter is as follows. The global scheme of the proposed heuristic is introduced in Section 3.1. The ALNS heuristic is detailed in Section 3.2. Our split algorithm is presented in Section 3.3. Computational results are presented in Section 3.4. Finally, we conclude by some remarks in Section 3.5.

3.1 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 [Prins et al., 2014]. 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 [Beasley, 1983] for the CVRP. Then, this method was incorporated within a genetic algorithm by Prins in [Prins, 2004].

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. [Vidal et al., 2015] and Vargas et al. [Vargas et al., 2017] 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. [Vidal et al., 2015] 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. [Vargas et al., 2017] 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 reffer the reader to [Prins et al., 2014].

Our solution method adopts also the *order first-cluster second* approach. Algorithm 3 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 $Eval(X)$ to denote the profit of a solution X. This process is iterated until one of the following stop conditions is reached : either a maximum number of iterations n is reached, where n is the number of customers, or a maximum number of iterations without improvement is exceeded, which is fixed at the average number of customers per cluster.

3.2 Adaptive large neighborhood search

The main feature of the ALNS is the use of multiple neighborhoods in parallel during the search process [Pisinger and Ropke, 2010]. These different neighborhoods are identified by a set of competing removal and insertion operators. 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 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

Algorithm 3: GLOBAL SCHEME

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

```
X_best ← X
LB ← Eval(X)
repeat
    ALNS(X)(see Section 3.2)
    Construct a giant tour GT from X
    X ← SPLIT(GT, LB)(see Section 3.3)
    if (Eval(X) > Eval(X_best)) then
        X_best ← X
        LB ← Eval(X)
until (stop condition is reached)
return X_best
```

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 because of the time limit 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. It calculates the gap in terms of travel time between the two best insertions of each customer. We call this gap *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 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 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 the best 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 [Pisinger and Ropke, 2010].

Algorithm 4: ALNS

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

$d_{max} \leftarrow 1 + rand()\%3$
$X_{best} \leftarrow X$
repeat
  Remove d_{max} clusters from X
  Apply 2-opt on X
  Select an insertion operator i
  Apply i on X
  if $(Eval(X) > Eval(X_{best}))$ **then**
    $X_{best} \leftarrow X$
    $d_{max} \leftarrow 1 + rand()\%3$
  else $d_{max} \leftarrow d_{max} + 1$

  Update weights using the adaptive weight adjustment procedure
until (*stop condition is reached*)
return X_{best}

3.3 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 a given giant tour $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ 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.

3.3.1 Preliminary result

In this subsection, we present a relaxation scheme for the CluTOP based on the TOP.

Definition 3.3.1. 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 profit of each customer in I_{TOP} is calculated as follows : $\rho_j = \sum_{k:j \in S_k} \frac{P_k}{|S_k|}$, where the ratio $\frac{P_k}{|S_k|}$ could be interpreted as the contribution of customer j to the cluster S_k. 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, $P_{TOP}(V') = \sum_{j \in V'} \rho_j$.

Proposition 3.3.1. *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 $P_{TOP}(V^*) = \sum_{j \in V^*} \rho_j$. We also denote the optimal objective value for I_{TOP} by $P^*(I_{TOP})$. We have,

$$\begin{aligned} P_{TOP}(V^*) = \sum_{j \in V^*} \rho_j &= \sum_{j \in V^*} \sum_{k:j \in S_k} \frac{P_k}{|S_k|} \\ &= \sum_{j \in V^*} \sum_{k:j \in S_k \text{ and } S_k \in S^*} \frac{P_k}{|S_k|} + \sum_{j \in V^*} \sum_{k:j \in S_k \text{ and } S_k \notin S^*} \frac{P_i}{|S_i|} \\ &= P_{CluTOP}(S^*) + \sum_{j \in V^*} \sum_{k:j \in S_k \text{ and } S_k \notin S^*} \frac{P_k}{|S_k|} \\ &= P^*(I) + \sum_{j \in V^*} \sum_{k:j \in S_k \text{ and } S_i \notin S^*} \frac{P_k}{|S_k|} \end{aligned} \tag{3.1}$$

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

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

Corollary 3.3.1. *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.*

3.3.2 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 as well as a feasibility test to discard unfeasible nodes. 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.

In what follows, we describe the different components implemented in our branch and bound algorithm.

3.3.3 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 $O(n)$.

3.3.4 Knapsack-based upper bound

We propose in this section an upper bound based on the Fractional Knapsack Problem (FKSP) and we take advantage of the cluster constraint in order to improve this upper bound.

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 3.3.1. 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 : $\rho_{\pi_j}^\eta = \sum_{i:\pi_j \in S_i \text{ and } S_i \in S_p^\eta} \frac{P_i}{|S_i^\eta|}$, where $|S_i^\eta|$ is the number of potential customers belonging to cluster S_i in 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 [Beasley, 1983] for VRP. In our study we use an effcient implementation proposed by Vidal in [Vidal, 2016] 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 3.3.2. *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 $\rho_{\pi_j}^\eta$.

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, 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 3.3.1, I_{FKSP} is also an upper bound on I_{CluTOP} while considering π and η. □

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^η 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 :

$$\nexists j'/(l < j' < j \ or \ j < j' < r) \quad and \quad \pi_{j'} \in S_s^\eta \tag{3.2}$$

- 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 (π_l, π_r or π_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 :

$$\begin{gathered} \nexists j'/(l < j' < j \ or \ j < j' < r) \\ and \quad (\Omega(\pi_l) \subseteq \Omega(\pi_{j'}) \ or \ \Omega(\pi_r) \subseteq \Omega(\pi_{j'}) \ or \ \Omega(\pi_j) \subseteq \Omega(\pi_{j'})) \end{gathered} \tag{3.3}$$

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

3.3.5 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 efficiently explore 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, which is depicted in Algorithm 5, 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 (line 3). It randomly selects in each iteration one unrouted cluster (line 5) 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 [Bouly et al., 2010]. If IDCH fails to insert the customers, the Lin-Kernighan TSP heuristic [Lin and Kernighan, 1973] is used (line 6).

Algorithm 5: ITERATIVE INSERTION

Input: Solution X
Output: Solution X

```
1  Δ ← unrouted clusters of X
2  insert ← true
3  while (Δ ≠ ∅  and  insert = true) do
4      insert ← false
5      foreach (k ∈ Δ) do
6          if (IDCH(X, k) = true) or (LinKernighan(X, k) = true) then
7              Δ ← Δ\{k}
8              insert ← true
9              break
10 return X
```

3.3.6 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 breadth-first search (BFS) and impose a limit on the number of nodes expanded in each level of the tree (See Algorithm 6). Consequently, this scheme does not guarantee that the solution found is optimal. Therefore, 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 (3.3.4) as a selection criteria. Another important aspect is the number of nodes selected at each level. This parameter was fixed after experimentation at K (the number of clusters) nodes per level.

Algorithm 6 describes the whole split procedure. We use in Algorithm 6 two ordered lists (line 1), one called *actList* that contains the nodes of the current level, whereas the second list *tmpList* contains the nodes of the next level. The lower bound LB is initialized by the best current best solution of the global heuristic.

Algorithm 6: SPLIT

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

Initialization : ordered list of the clusters $Order$ used as branching strategy, current level $L \leftarrow 1$, current node $e \leftarrow rootNode$, $actList \leftarrow e, tmpList \leftarrow \emptyset$
while *($actList \neq \emptyset$ and $L \leq K$)* **do**
    Select the best node e in $actList$ based on Knapsack UB (See Section 3.3.4)
    Expand e to two nodes e_1 and e_2 based on $Order(L)$ (See Section 3.3.2)
    foreach $(e \in \{e_1, e_2\})$ **do**
        if (e *is infeasible*) **then continue** (See Section 3.3.3)
        if (*Knapsack UB of* $(e) \leq LB$) **then continue** (See Section 3.3.4)
        $tmpList \leftarrow tmpList \cup \{e\}$
        Extract solution X from e
        Apply LS on X (See Section 3.3.5)
        if $(Eval(X) > Eval(X_{best}))$ **then**
            $X_{best} \leftarrow X$
            if $(Eval(X) > LB)$ **then** $LB \leftarrow Eval(X)$
    if $(actList = \emptyset)$ **then**
        $actList \leftarrow tmpList$
        $tmpList \leftarrow \emptyset$
    $L++$
Select the best node e in $actList$ and Extract solution X_{best}
(See Section 3.3.4)
return X_{best}

3.4 Performance of the heuristic method

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 [Angelelli et al., 2014]. 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 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. [Angelelli et al., 2014].

3.4.1 Single vehicle

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 [Angelelli et al., 2014]. Three variants of COP-TABU were implemented : COP-TABU-*Basic*, COP-TABU-*Multistart* and COP-TABU-*Reactive*. Tests were conducted on all the 57 classes of the benchmark [Angelelli et al., 2014] (924 instances). We ran our algorithm 10 times per instance and we recorded the best solution found.

Table 3.1 summarizes the obtained results. In this table, for each method, we provide :

- $\#BEST$: The number of times it yields the best known solution.
- RPE : Average percentage error. The RPE for each instance is calculated using the following expression :

$$RPE = \frac{Z_{best} - Z_{max}}{Z_{best}} \times 100 \tag{3.4}$$

  where Z_{best} is the best solution found among the 10 runs and Z_{max} is the best solution in the literature, including our method.

- CPU : The average CPU time.

Tableau 3.1 – Performance of the Hybrid Heuristic

Class	COP-TABU-*Basic*			COP-TABU-*Multistart*			COP-TABU-*Reactive*			Hybrid Heuristic		
	$\#BEST$	RPE	CPU	$\#BEST$	RPE	CPU	$\#BEST$	RPE	CPU	$\#BEST$	RPE	CPU
dantzig42	16	0.000	13.27	16	0.000	17.77	16	0	38.95	16	0	2.42
swiss42	13	0.719	15.38	14	0.281	23.09	15	0.013	31.93	**16**	0	1.56
att48	16	0.000	18.24	16	0.000	26.08	15	0.062	38.76	16	0	5.44
gr48	11	5.709	13.74	12	3.184	26.02	16	0	37.96	16	0	2.06

continued on next page

Table 3.1 – continued from previous page

Class	COP-TABU-*Basic*			COP-TABU-*Multistart*			COP-TABU-*Reactive*			Hybrid Heuristic		
	#BEST	*RPE*	*CPU*	*#BEST*	*RPE*	*CPU*	*#BEST*	*RPE*	*CPU*	*#BEST*	*RPE*	*CPU*
*hk*48	15	1.250	20.58	16	0.000	30.76	15	0.315	37.68	16	0	3.08
*eil*51	11	2.181	15.92	11	2.181	24.46	15	0.242	36.83	15	0.059	2.71
*berlin*52	15	0.548	38.88	15	0.120	53.39	15	0.120	60.41	**16**	0	6.55
*brazil*58	13	0.573	58.99	14	0.115	75.72	16	0	83.97	16	0	12.58
*st*70	11	1.303	23.18	11	1.012	38.95	12	0.639	48.01	**16**	0	5.09
*eil*76	9	6.407	24.50	10	4.050	33.74	15	0.125	45.84	**16**	0	4.04
*pr*76	11	1.014	21.40	13	0.105	30.88	15	0.009	54.76	**16**	0	8.82
*gr*96	12	0.612	44.07	13	0.116	51.35	14	0.025	68.19	**16**	0	9.24
*rat*99	12	1.752	32.99	12	0.127	52.03	15	0.034	63.65	**16**	0	11.48
*kroA*100	11	6.013	44.65	14	0.123	50.98	14	0.429	52.62	**16**	0	6.22
*kroB*100	15	0.714	47.96	16	0.000	58.94	16	0	62.20	16	0	7.04
*kroC*100	10	3.687	37.55	15	0.269	48.74	14	0.452	59.42	**16**	0	7.35
*kroD*100	10	1.879	36.85	11	1.247	56.70	13	0.520	69.57	**16**	0	9.08
*kroE*100	12	2.889	46.59	12	1.374	48.83	14	0.270	62.77	**16**	0	6.47
*rd*100	12	1.431	36.51	13	1.030	47.81	15	0.568	82.29	**16**	0	8.02
*eil*101	7	2.495	32.97	12	0.729	44.62	**16**	0	79	15	0.030	9.46
*lin*105	11	1.393	36.06	13	0.461	52.48	14	0.348	105.21	**16**	0	20.80
*pr*107	13	6.350	72.19	15	0.203	86.35	15	0.160	135.39	**16**	0	60.38
*gr*120	10	2.917	50.87	11	2.856	66.36	14	0.185	105.25	**16**	0	17.78
*pr*124	14	1.180	80.33	16	0.000	88.26	16	0	150.15	16	0	25.90
*bier*127	12	0.873	63.05	14	0.108	94.57	15	0.005	149.64	**16**	0	23.89
*ch*130	7	4.016	49.79	9	2.949	64.58	12	1.376	106.57	**16**	0	13.88
*pr*136	12	1.588	59.86	14	0.949	71.37	15	0.694	121.5	**16**	0	17.50
*gr*137	15	0.156	82.07	16	0.000	104.45	16	0	181.54	16	0	19.93
*pr*144	16	0.000	168.25	16	0.000	175.28	16	0	247.29	16	0	39.90
*ch*150	8	2.684	34.19	8	2.543	53.97	14	0.554	101.37	**16**	0	19.29
*kroA*150	9	1.002	36.84	13	0.228	50.6	14	0.074	102.11	**16**	0	18.40
*kroB*150	8	2.456	40.06	10	2.127	56.69	14	0.621	107.93	**16**	0	15.27
*pr*152	15	0.545	120.08	16	0.000	164.81	16	0	248.1	16	0	38.57
*u*159	6	3.300	113.36	9	2.373	125.51	8	1.447	184.68	**16**	0	54.53
*si*175	16	0.000	47.69	16	0.000	63.36	16	0	126.80	16	0	274.37
*brg*180	12	0.656	54.29	13	0.578	72.18	15	0.091	127.74	**16**	0	177.20
*rat*195	12	0.531	68.18	10	0.209	78.52	14	0.401	172	**16**	0	45.40
*d*198	15	0.062	172.56	16	0.000	217.29	16	0	368.98	16	0	43.81
*kroA*200	11	1.130	55.09	12	1.093	76.17	14	1.052	139.43	**16**	0	33.13
*kroB*200	8	2.610	71.45	10	1.978	87.73	13	0.129	142.43	**16**	0	32.43
*gr*202	11	1.256	88.17	12	1.001	121.27	16	0	236.24	16	0	47.86
*ts*225	12	0.259	162.94	12	0.158	189.13	15	0.019	234.81	**16**	0	61.22
*tsp*225	9	1.583	87.78	9	0.495	102.99	11	0.142	180.26	**16**	0	59.03
*pr*226	12	0.872	244.84	12	0.787	268.33	15	0.042	331.10	**16**	0	96.38
*gr*229	15	0.023	109.99	15	0.023	121.07	15	0.023	170.85	**16**	0	30.45
*gil*262	7	8.107	57.09	6	4.296	84.20	10	2.441	135.48	**15**	0.032	63.58
*pr*264	11	4.230	151.96	10	4.243	208.51	14	0.323	304.70	**16**	0	79.42
*a*280	11	0.159	99.98	12	0.156	150.54	10	0.255	191.39	**14**	0.029	199.93
*pr*299	10	1.097	105.14	10	1.089	125.98	12	0.614	205.23	**16**	0	184.52
*lin*318	8	1.009	247.01	9	0.870	260.81	11	0.492	311.25	**16**	0	234.81
*rd*400	9	2.387	100.44	10	1.808	147.13	11	1.784	203.08	**16**	0	203.84
*fl*417	11	1.055	518.97	12	0.397	577.53	13	0.079	708.57	**16**	0	194.10
*gr*431	12	0.788	236.75	15	0.009	252.35	16	0	280.53	16	0	152.84
*pr*439	11	0.685	180.16	13	0.074	221.23	14	0.058	324.17	**16**	0	202.01
*pcb*442	12	0.331	151.28	11	0.549	199.82	13	0.532	274.18	**16**	0	306.97
*d*493	7	1.157	418.35	9	1.208	419.66	12	1.051	515.84	**16**	0	362.25

continued on next page

Table 3.1 – continued from previous page

Class	COP-TABU-*Basic*			COP-TABU-*Multistart*			COP-TABU-*Reactive*			Hybrid Heuristic		
	#BEST	*RPE*	*CPU*	*#BEST*	*RPE*	*CPU*	*#BEST*	*RPE*	*CPU*	*#BEST*	*RPE*	*CPU*
*att*532	17	1.725	1707.99	20	1.415	2180.37	25	0.888	2244.39	25	0.074	1848.53
Total	657	1.499	153.719	720	0.843	174.37	816	0.328	223.88	**916**	0.005	118.36

On the basis of Table 3.1, we observe that our heuristic algorithm outperforms all COP-TABU versions. It delivers 82 new BKS to reach 916 BKS. On the other hand the three tabu versions heuristics yield 656, 720 and 816 BKS. Moreover, our heuristic exhibits a very small relative percentage error of 0.005 compared to 0.328 for COP-TABU (*Reactive*). Finally, we observe that our method improves the results of several classes of instances namely : $ch150$, $u159$, $tsp225$, $gil262$, $a280$, ...

To have more insight, we propose in the following to study the impact of the number of clusters on the performance of the heuristic methods. We note that the profit generation pattern and the value of θ did not reflect any evident impact on the behavior of neither COP-TABU [Angelelli et al., 2014] nor the hybrid heuristic. We compare in a first step the heuristic results with the optimal solutions and the best upper bounds obtained by the exact methods (the cutting plane of the branch-and-cut algorithms).

Figures 3.1 and 3.3 show the heuristics' results related to instances with up to 318 vertices (800 small instances) compared with the results of exact methods presented in [Angelelli et al., 2014] and Chapter 2. Figures 3.1 reports the average gap with respect to the upper bound while figure 3.3 illustrates the gap to the optimal solution while considering optimally solved instances only. According to figure 3.1, the hybrid heuristic achieved an average gap below 1.2%, whilst the best tabu variant, the COP-TABU-*Reactive*, has an average gap below 1.7%. The other COP-TABU variants achieved relatively higher gaps with less than 3.4% for COP-TABU-*Basic* and less than 2.2% for COP-TABU-*Multistart*. It is noteworthy to mention that the gaps of COP-TABU are less than those reported in [Angelelli et al., 2014], which were about 4.8% for COP-TABU-*Reactive*. This is due mainly to the substantial improvement made by the cutting plane, and hence, gaps to the upper bound are more related to the performance of the heuristic methods rather than the quality of the upper bound.

Regarding the gap to the optimal solution, our heuristic achieved almost 0% on instances with up to 318 vertices, while COP-TABU-*Reactive* succeeded to have a gap less than 0.3%. More in detail, among 679 instances solved to optimality by all the exact methods, the hybrid heuristic finds 677 optimal solutions, COP-TABU-

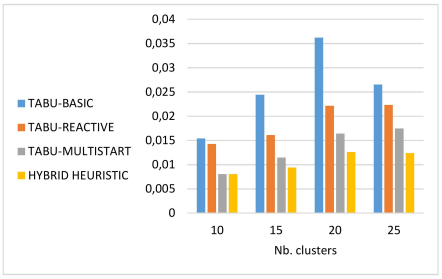


Figure 3.1 – Gap with respect to the upper bound

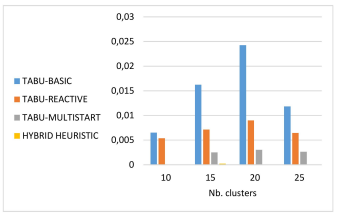


Figure 3.3 – Gap with respect to the optimal solution

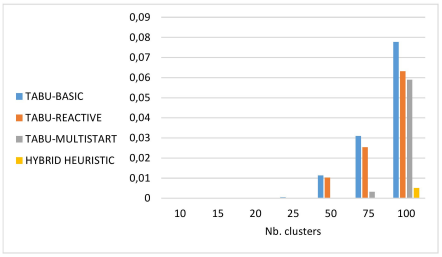


Figure 3.2 – Gap with respect to the best solution found

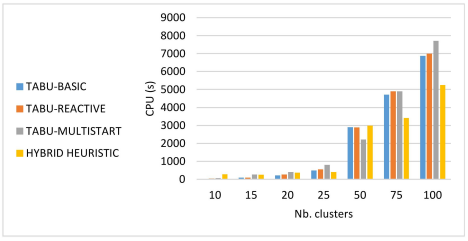


Figure 3.4 – Computational time

Reactive finds 626, COP-TABU-*Multistart* and COP-TABU-*Basic* find 565 and 514 respectively.

Figures 3.2 and 3.4 illustrate a comparison between all the heuristics while considering all the benchmark instances. Figure 3.2 depicts the average error with respect to the best solution found by the four heuristics. We notice that the hybrid heuristic succeeded to maintain a gap close to 0% regardless of the number of clusters, except for $K = 100$, where the gap did not exceed 0.5%, whereas COP-TABU-*Reactive*, which the best COP-TABU variant, achieved a gap of 5.9% for the same number of clusters. Results concerning computational time depicts the same behavior for all the heuristics : the more the number of clusters, the more are computational times. We notice also a slight advantage for our heuristic when the number of clusters goes up.

3.4.2 Multiple vehicles

In this last section, we discuss the performance of our proposed heuristic in the case of multiple vehicles. In Table. 3.2, 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 : Average CPU time ;
- AVG : the error gap between the average solution value and the best solution value among the ten executions of each instance.

The results depicted in Table 3.2 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 can also observe a slight increase in the value of AVG when the number of vehicles increases but still has small values (2.1% for three vehicles and 0.3% 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 close to 0. In addition, the heuristic method succeeds in finding almost all the optimal solutions found by the exact method : 400/405 with two vehicles, 496/502 with three vehicles and 677/679 in the case of a single vehicle.

Tableau 3.2 – Multiple vehicles

	$UBGap$	$OptGap$	$\#Opt$	CPU	AVG
1 vehicle	0.8%	$\approx 0\%$	677/679	118.46s	0.3%
2 vehicles	10.1%	0.03%	333/337	108.4s	1.06%
3 vehicles	11.61%	0.55%	459/463	109.65s	2.49%

Figures 3.5, 3.6 and 3.7 depict the performance of the heuristic with respect to the number of vehicles and the number of clusters. According to fig. 3.5, 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

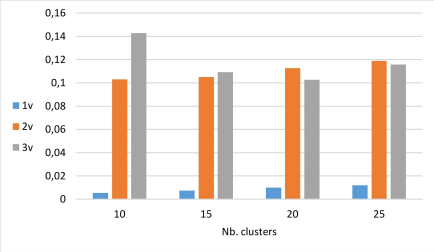


Figure 3.5 – Average gap to the best upper bound.

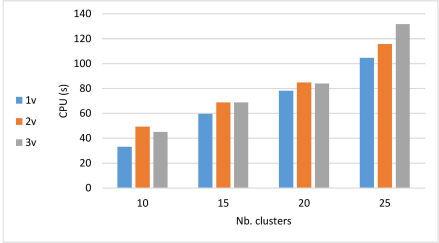


Figure 3.6 – Computational times with respect to the number of clusters.

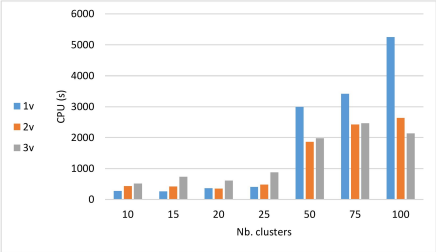


Figure 3.7 – Computational times with respect to the number of clusters on instances with 532 vertices.

behavior of the heuristic for instances with up to 25 clusters (fig. 3.6). 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. 3.7. 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.

3.5 Conclusion and future work

In this chapter, 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. As future work, our aim is to propose different extensions for the CluTOP. Also, additional constraints like time windows or vehicle capacity should be considered.

GRASP×ILS with Constraint Programming for a Synchronized Team Orienteering Problem with Time Windows

Sommaire

4.1 Introduction

Wildfires have become in the last decade a frequent phenomenon causing important damages to private properties, community assets as well as human life. Many countries have witnessed the devastating impact of wildfires on the nature and human activities. Asset protection activities performed by Incident Management Teams (IMT) during wildfires are therefore of a crucial importance in order to minimize risks of losing vital infrastructure. However, several challenging tasks and difficulties complicate the working environment of IMT's in which they must make critical and complex decisions. Thus, the application of operations research techniques, like mathematical programming and heuristics, can enhance the management of wildfires in such hostile situations.

We consider the case of an out of control wildfires spreading across a landscape and threatening a number of assets like bridges, electric substations, schools and factories. Defensive activities carried out by IMT near of assets before fire impact are important to reduce the risk of losing them. Examples of defensive tasks are removing debris and combustible materials, wetting down buildings or putting out fires. To be successful, IMT activities should take action before fire fronts reach endangered assets, but not too early otherwise the intervention would be ineffective. To that end, fire progression can be estimated by using meteorological data and fire propagation models. Moreover, some assets may require the intervention of several trucks and equipment with specific capabilities. These trucks should collaborate together in a timely manner to carry out protection activities.

Such situation can be modeled using vehicle routing problems with additional constraints such as time windows and synchronized visits. A fleet of heterogeneous vehicles is available to visit a set of assets. Each asset is associated with a time window representing fire fronts progression. An asset is also assigned a service time duration which models the time necessary to perform the protection activities. Finally, each asset has a resource requirements which are expressed by the number and type of required vehicles. Due to these constraints, protecting all the assets might be impossible. Hence, a value, called profit, is associated with each asset in order to distinguish between different assets according to their relative importance. In order to gain the profit of a given asset, it must be visited by the required resources in a synchronized manner, i.e. the visits should be performed simultaneously and cooperatively by the vehicles within the corresponding time window. As a result, the objective function aims at maximizing the amount of profit collected. In the rest of the paper, we denote this problem as the Synchronized Team Orienteering Problem with Time Windows (STOPTW).

The STOPTW was first proposed by [van der Merwe et al., 2014] under the Asset Protection Problem During Escaped Wildfire. The authors introduced a mixed integer programming model for the asset protection problem during escaped wildfires, which was demonstrated on a realistic wildfire scenario in Tasmania. [Roozbeh et al., 2018] proposed an Adaptive Large Neighborhood Search Heuristic (ALNS) for the problem along with new set of benchmark instances. [van der Merwe et al., 2017] developed a dynamic approach to reroute vehicles during firefighting once disruptions occur. The method aims at maximizing the total value of protected assets while minimizing the number of changes on rescue plans elaborated earlier.

In this paper, we propose a new constraint programming model for the STOPTW. The CP model succeeds to solve to optimality all the small instances.

For medium-size instances, the CP model finds very competitive results compared to the ALNS method proposed in [Roozbeh et al., 2018]. We also propose a Greedy Randomized Adaptive Search Procedure (GRASP) coupled with an Iterated Local Search (ILS) as a heuristic method. The GRASP×ILS incorporates an insertion method based on a constraint programming model which is used to repair partially destroyed solutions. A post-optimization module based on a set covering formulation is used to extract the best solution from a pool of feasible tours.

The remainder of this paper is organized as follows. The mathematical formulation of the problem is given in Section 4.2. A new formulation that uses only binary and real decision variables is provided in Section 4.3. The Constraint Programming model is described in Section 4.4. The GRASP×ILS method is introduced in Section 4.5. Computational tests carried out on the methods proposed in this paper are extensively described in Section 4.6. Finally, a conclusion and some perspectives are given in Section 4.7.

4.2 Problem description and mathematical formulation

A mixed integer programming model (MIP) for the STOPTW was proposed in [van der Merwe et al., 2014]. Let $G = (V, A)$ be a directed graph where $V = \{0, 1, \ldots, n+1\}$ is the vertex set and $A = \{(i, j) : i, j \in V, i \neq j\}$ is the arc set. The set $V^- = \{1, \ldots, n\}$ represents the set of assets to be protected whereas 0 and $n+1$ represent the departure and arrival depots. For convenience, we define two index sets $V^d = \{0, 1, \ldots, n\}$ and $V^a = \{1, 2, \ldots, n+1\}$. A fleet with different vehicle types $q \in Q$ is used to serve the assets with P_q vehicles of each type q are available. A cost t_{ijk} is used to model the travel time necessary for a vehicle of type $q \in Q$ to traverse arc $(i, j) \in A$. Each asset $i \in V^-$ is associated with the following data :

- a time window $[o_i, c_i]$, where o_i represents the earliest service time and c_i the latest service time.
- a resource requirements vector $R_i =< r_{i1}, r_{i2}, \ldots, r_{i|Q|} >$, where r_{iq} is the number of vehicles of type q $(1 < q < |Q|)$ required by asset i.
- a service duration a_i, which is the time needed to protect asset i during wildfire.
- a profit p_i that represents the value of asset i.

Before detailing the mathematical programming model, we first introduce the decision variables.

- y_i : binary decision variable, it takes the value 1 if vertex i is protected, 0 otherwise.
- s_i : real decision variable in $[o_i, c_i]$ associated with each vertex $i \in V$, at which the service must start in order to protect asset i and gain its profit.
- z_{ijq} : binary decision variable, equal to 1 if the arc (i, j) is traversed by at least one vehicle of type q, 0 otherwise.
- x_{ijq} : integer decision variable, it indicates the number of vehicles of type q that traversed the arc (i, j).

We introduce in the following the mathematical formulation of the problem :

$$max \sum_{i=1}^{n} p_i y_i \tag{4.1}$$

$$\sum_{i \in V^a} x_{0iq} = \sum_{j \in V^d} x_{j,n+1,q} = P_q \quad \forall q \in Q \tag{4.2}$$

$$\sum_{j \in V^d} x_{jiq} = \sum_{h \in V^a} x_{ihq} \quad \forall q \in Q, \forall i \in V^- \tag{4.3}$$

$$\sum_{j \in V^d} x_{jiq} = R_{iq} y_i \quad \forall q \in Q, \forall i \in V^- \tag{4.4}$$

$$x_{ijq} \leq P_q z_{ijq} \quad \forall q \in Q, \forall (i,j) \in A \tag{4.5}$$

$$z_{ijq} \leq x_{ijq} \quad \forall q \in Q, \forall (i,j) \in A \tag{4.6}$$

$$s_i + t_{ijq} + a_i - s_j \leq M(1 - z_{ijq}) \quad \forall q \in Q, \forall (i,j) \in A \tag{4.7}$$

$$o_i \leq s_i \leq c_i \quad \forall i \in V \tag{4.8}$$

$$x_{ijq} \in \{0, 1, 2, \ldots, P_q\} \quad \forall q \in Q, \forall (i,j) \in A \tag{4.9}$$

$$y_i \in \{0, 1\} \quad \forall i \in V^- \tag{4.10}$$

$$z_{ijq} \in \{0,1\} \quad \forall q \in Q, \forall (i,j) \in A \tag{4.11}$$

The objective function (4.1) is to maximize the total collected profit. Constraints (4.2) ensure that all the P_q available vehicles start from the departure depot and end at the arrival depot. Constraints (4.3) impose that the number of incoming and outgoing vehicles is the same at each asset and for each vehicle type. Constraints (4.4) ensure that if a customer is served, then all of its requirements in terms of number of vehicles are met. Constraints (4.5) limit the capacity of each arc of type q to at most P_q vehicles. Constraints (4.6) are coupling constraints between z and x variables. Constraints (4.7) guarantee the connectivity of each tour whereas constraints (4.8) are the time windows constraints. Constraints (4.9),(4.10) and (4.11) are domain definitions.

4.3 New formulation

The mathematical model provided in previous section contains, in addition to real and binary decision variables, an integer variable z. We propose in this section a mathematical formulation that uses real and binary decision variables only. Let us first consider an auxiliary directed graph $H = (X, A)$, where $X = \{0, 1, \ldots, N, N{+}1\}$ is the set of vertices with 0 and $N + 1$ are, respectively, the departure and the arrival depots, and A is the set of arcs. Each asset is represented in H by a set of superposed vertices, whose the number depends on the resources required by this asset. For instance, if a given asset needs one vehicle of type $q \in Q$ and one vehicle of type $q' \in Q$, two superposed vertices will be associated with it. As a result, the set of vertices X is the union of $|Q|$ disjoint sets $X = \cup_{q \in Q} X_q$, where each set X_q contains the vertices related to the type $q \in Q$ of vehicles. We denote by X_q^-, X_q^d and X_q^a, respectively, the set of vertices of type $q \in Q$ without the depots, the set of vertices of type $q \in Q$ without the arrival depot and the set of vertices of type $q \in Q$ without the departure depot. We denote by $Sync_j$ the set of vertices associated with asset $i \in V$, whereas $Sync_j^{-1}$ is used to denote the asset associated with the vertex $j \in X$. We also use the additional following decision variables described as follows :

- u_j binary variable equal to 1 if vertex j is visited, 0 otherwise.
- w_{ij} binary variable equal to 1 if the arc (i, j) is traversed by a vehicle, 0 otherwise.

$$max \sum_{i=1}^{n} p_i y_i \tag{4.12}$$

$$\sum_{i \in X_q} w_{0i} = \sum_{j \in X_q} w_{j,N+1} = P_q \quad \forall q \in Q \tag{4.13}$$

$$\sum_{j \in X_q^d} w_{ji} = \sum_{h \in X_q^a} w_{ih} \quad \forall i \in X_q, \forall q \in Q \tag{4.14}$$

$$\sum_{j \in X_q^d} w_{ji} = u_i \quad \forall i \in X_q \quad \forall q \in Q \tag{4.15}$$

$$u_j = y_i \quad \forall j \in Sync_i, \forall i \in V \tag{4.16}$$

$$s_{Sync_i^{-1}} + t_{ij} + a_i - s_{Sync_j^{-1}} \leq M(1 - w_{ij}) \quad \forall i, j \in X_q | (i,j) \in A \quad \forall q \in Q \tag{4.17}$$

$$o_i \leq s_i \leq c_i \quad \forall i \in V \tag{4.18}$$

$$y_i \in \{0,1\} \quad \forall i \in V \tag{4.19}$$

$$u_i \in \{0,1\} \quad \forall i \in X \tag{4.20}$$

$$w_{ij} \in \{0,1\} \quad \forall i, j \in X_q | (i,j) \in A \quad \forall q \in Q \tag{4.21}$$

The objective function (4.12) aims to maximize the total collected profit. Constraints (4.13) impose a limit on the number of vehicles of each type $q \in Q$, whereas flow conservation constraints are imposed by (4.14). Constraints (4.15) along with (4.16) guarantee the respect of resource requirements in case an asset is served. Temporal constraints are assured by (4.17) and (4.18). Domain definitions are described by (4.19-4.21).

4.4 Constraint programming model

We propose in the following a constraint programming (CP) based model for the STOPTW. CP is a powerful paradigm able to find good quality solutions in short computational times. The key feature of CP is the mechanism of constraint propagation which is used to filter out variable values that cause infeasibility. More specifically, complex relationships between variables are expressed using

global constraints. These constraints are associated with effective filtering domain algorithms, which are triggered whenever a change occurs in the domain of a shared variable. The aim is to propagate the new information to other constraints so that domain values of other decision variables can be reduced.

This paradigm has been widely used to solve scheduling and timetabling problems. However, few works that use CP for routing problems have been published. Gedik et al. [Gedik et al., 2017] proposed a CP model for the Team Orienteering Problem with Time Windows (TOPTW), in which they tested different branching strategies. The CP model succeeded to solve to optimality a large number of instances and found a new optimal solution for one benchmark instances. Rousseau et al. [Rousseau et al., 2013] proposed a flexible heuristic based on a CP model to solve an online variant of vehicle routing problem with synchronized visits called the Synchronized Dynamic Vehicle Dispatching Problem (SDVDP). The same CP model was used to solve the offline version of the problem in [Hojabri et al., 2018].

The proposed CP model is based on interval decision variables. This type of variables, initially designed to solve scheduling problems, is capable of representing several critical decisions such as start times, services/jobs duration and end times in a single decision variable [IBM, 2018]. This characteristic allows to considerably reduce the number of decision variables compared to the traditional mathematical programming. Another important characteristic of interval variables is their inherent boolean behavior. By declaring an interval variable as optional, it may or it may not appear in the final solution. This feature is very useful in order to model assignment decisions in scheduling problems or the selectivity aspect in team orienteering problems. In addition, binary relationships that involve optional interval variables are systematically discarded when one of the sides is absent in the solution, which avoids the need to additional decision variables or constraints.

We present in the following some necessary notation and decision variables used in the CP model.

- ast_i : an optional interval variable associated with asset $i \in V^-$. This interval variable is associated with the earliest start time o_i, the latest end time $c_i + a_i$ and the service time duration a_i. If asset i is present in the optimal solution, ast_i provides then its starting service time s_i within $[o_i, c_i]$.

- vst_{iqp} : an interval variable which indicates whether the asset $i \in V$ is visited by vehicle $p \in \{1, \ldots, P_q\}$ of type $q \in Q$. Variables vst_{nqp} and $vst_{(n+1)qp}$ are mandatory and they represent the departure and the arrival depots, whereas interval variables vst_{iqp}, $i \in V^-$ are optional. We set for each $i \in V^-$ an earliest start time o_i and a latest end time $c_i + a_i$. If $i = n$, the latest start

time is set to 0.

- $vhcl_{qp}$: an interval sequence variable defined on the set S_{qp} of interval variables where $S_{qp} = \{vst_{0qp}, \ldots, vst_{(n+1)qp}\}$, $p \in \{1, \ldots, P_q\}$, $q \in Q$. The variable $vhcl_{qp}$ indicates the assets visited by vehicle $p \in \{1, \ldots, P_q\}$ of type $q \in Q$ as well as the ordering of these visits. Absent interval variables are not considered in the ordering.

- $G_{iq} = \{vst_{iq1}, vst_{iq2}, \ldots, vst_{iqP_q}\}$: a set of interval variables that contains the possible vehicles of type $q \in Q$ that can visit the asset $i \in V^-$.

Despite the several initiatives launched over the past years, there is no standardized formalism for CP. As a result, we adopted a syntax close to that of the implementation. In addition, some decision variables as well as some global constraints used in our model are extracted from IBM's CP Optimizer solver and may not be provided by other solvers such as Gecode.

$Alternative(b_0, \{b_1, b_2, \ldots, b_n\}, k)$ global constraint establishes an exclusive relationship among interval variables $\{b_1, b_2, \ldots, b_n\}$. If interval variable b_0 is present, then exactly k interval variables in $\{b_1, b_2, \ldots, b_n\}$ must be present. Moreover, the selected variables must start and end at the same time as b_0.

$NoOverlap(B, TansitionDistance(.))$ global constraint assures that the interval variables in the interval sequence variable B must be ordered in a such way that there is no overlapped variables, i.e. if b_i precedes b_j in B, then b_i must end before b_j starts. In addition, the function $TransitionDistance(.)$ imposes a minimum distance between every couple of interval variables in B present in the final solution.

The CP model for STOPTW, denoted by CP-STOPTW, is as follows :

$$max \sum_{i \in V^-} p_i \cdot PresenseOf(ast_i) \tag{4.22}$$

$$Alternative(ast_i, G_{iq}, r_{iq}) \quad \forall i \in V^-, \forall q \in Q \tag{4.23}$$

$$NoOverlap(vhcl_{qp}, TransitionDistance(t_{ij} | i \in V^d, j \in V^a)) \; \forall q \in Q, \forall p \in \{1, \ldots, P_q\} \tag{4.24}$$

$$vhcl_{qp}.First(vst_{nqp}) \quad \forall q \in Q, \forall p \in \{1, \ldots, P_q\} \tag{4.25}$$

$$vhcl_{qp}.Last(vst_{(n+1)qp}) \quad \forall q \in Q, \forall p \in \{1, \ldots, P_q\} \tag{4.26}$$

The objective function is to maximize the total collected profit (4.22). The function *PresenceOf* is used to verify whether the interval variable in argument is present in the final solution or not. Constraints (4.23) guarantee that the resource requirements are satisfied in terms of number and type of vehicles. At the same time, they assure the synchronization between different vehicles visiting the same asset using the global constraint *Alternative*. Constraints (4.24) guarantee that the routes assigned to vehicles are feasible in terms of time windows and travel times through the global constraint *NoOverlap*. $TransitionDistance(i, j)$ return the minimal travel time between every couple of assets. These constraints help also to prevent the existence of sub-tours in the final solution. Constraints (4.25) and (4.26) impose that the vehicles should start and end at the depot.

4.5 GRASP×ILS

Greedy Randomized Adaptive Search Procedure (GRASP) is a multi-start local search metaheuristic introduced by Feo and Resende in [Feo and Resende, 1995]. In each iteration, a new solution is generated using a greedy randomized heuristic. A local search procedure is then applied in order to improve the current solution. The best solution is recorded and updated each time a new best solution is found. The Iterated Local Search (ILS) is a heuristic scheme introduced in [Lourenço et al., 2003]. The basic idea of this method is to construct in each iteration a new solution using an embedded heuristic, but instead of starting each time from scratch or from a random solution, the embedded heuristic uses the solution of the previous iteration. The series of locally optimal solutions produced by this process can be seen as a single chain followed by the ILS. In GRASP×ILS, the local search phase in GRASP is replaced by the ILS in order to diversify the search and cover a larger search space. GRASP×ILS was successfully applied on many vehicle routing problems such as two-echelon location-routing problem [Nguyen et al., 2012], the periodic VRP with time windows [Michallet et al., 2014], and recently applied on a VRP with synchronization and precedence constraints [Haddadene et al., 2016]. In this section, we present our GRASP×ILS global framework to solve the STOPTW. Two ILS stages are embedded inside the GRASP, the first stage is based on a heuristic insertion method based on a candidate list (see Section 4.5.2). In the second stage ILS, a CP based insertion is used to repair partially destroyed solutions (see Section 4.5.3). At the end of each iteration of the GRASP, a post-optimization phase based on a set cover formulation is launched on a set of routes generated during ILS stages. The aim is to find the best combination that maximizes the total collected profit.

4.5.1 General flow

The key feature of our method is the use of two different insertion algorithms inside the GRASP. The first method is heuristic. It constructs the solution in a greedy fashion by iteratively inserting unrouted assets one by one based on a candidate list. The second approach is based on constraint programming, which tries to insert the maximum of assets in an exact fashion having as global objective the maximization of the total collected profit. CP-based construction proved to be very effective especially in routing problems with synchronization constraints [Hojabri et al., 2018, Rousseau et al., 2013]. Although the CP-based insertion is expensive in terms of computational time, it allows the ILS to cover larger neighborhood than the heuristic approach. Hence, a good compromise between the two methods would be interesting. The GRASP×ILS metaheuristic is sketched in Algorithm 7. The outer loop describes the structure of the GRASP in which $IterMax_G$ initial solutions are generated from scratch using candidate list-based insertion algorithm, described in Section 4.5.2. Each initial solution S is then improved using two blocks of iterative local search ILS. The first ILS block (lines 5-14) incorporates a perturbation phase, which partially destroys the current solution, and a repair phase performed by the candidate list-based insertion. Each time a new best solution is found, S_{best} is updated. The process is completed after $IterMax_{CL}$ iteration without improvement. The second ILS block (lines 18-25) starts also by removing a subset of assets from the solution. The repair phase is performed by the CP-based insertion described in Section 4.5.3. This process is repeated until a new best solution is found or $IterMax_{CP}$ iterations are expired. After the two ILS are completed, the global best solution S^* is updated.

A suitable perturbation technique is necessary for the ILS in order to improve the quality of its solutions. To that end, in each iteration of the ILS, a number of assets are randomly selected and removed from the current solution. The perturbation parameter (d_{CL} or d_{CP}) is initialized to 3 and is incremented after each iteration without improvement. Once a new best solution is found, the perturbation parameter is reset to 3. The number of assets to remove is important for the overall performance of the heuristic. When it has small values, it allows the ILS to explore the close neighborhood of the passed solution. On the other hand, when it has a large value, it gives the ability to the ILS to escape from local optima.

4.5.2 Candidate list-based insertion

The main component of ILS is the insertion algorithm. This algorithm starts from an initial solution, that can be empty, and add unrouted assets one by one. The

Algorithm 7: GRASP×ILS

Input: *Solution* S^*

for $i \leftarrow 1$ ***to*** $IterMax_G$ **do**
  $S \leftarrow CandidateListInsertion(S)$
  $S_{best} \leftarrow S$
  $Iter_{CL} \leftarrow 0$
  while $Iter_{CL} < IterMax_{CL}$ **do**
    $d_{CL} \leftarrow \mathcal{U}(1, d_{max})$ Remove d assets from S
    $S \leftarrow CandidateListInsertion(S)$ (see Section 4.5.2)
    if $(Profit(S) > Profit(S_{best}))$ **then**
      $S_{best} \leftarrow S$
      $d_{CL} \leftarrow 3$
      $Iter_{CL} \leftarrow 0$
    else
      $d_{CL} \leftarrow d_{CL} + 1$
      $Iter_{CL} \leftarrow Iter_{CL} + 1$
  $S \leftarrow S_{best}$
  $Iter_{CP} \leftarrow 0$
  $NoImpr \leftarrow true$
  while $(Iter_{CP} < IterMax_{CP})$ *and* $(NoImpr = true)$ **do**
    Remove d_{CP} assets from S
    $S \leftarrow CPBasedInsertion(S)$ (see Section 4.5.3)
    if $(Profit(S) > Profit(S_{best}))$ **then**
      $S_{best} \leftarrow S$
      $NoImpr \leftarrow false$
    else $d_{CP} \leftarrow d_{CP} + 1$
    $Iter_{CP} \leftarrow Iter_{CP} + 1$
  $S_c \leftarrow setCover()$ (see Section 4.5.4)
  if $(S_c > S_{best})$ **then** $S_{best} \leftarrow S_c$
  if $(S_{best} > S^*)$ **then** $S^* \leftarrow S_{best}$
return S^*

insertion process stops when all the assets are inserted or no more insertions are possible. The order of insertions is based on a preceding sorting of assets according to non-decreasing values of a specific criterion. We consider in the insertion criterion the following factors :

- Since the objective function is to maximize the collected profit, this criterion favors the assets with higher values of profit to be inserted.
- We also consider in the sorting criterion the width of the time windows. Intuitively, assets with large time windows are likely more flexible to insert. Thus, assets with tight time windows have the priority to be inserted first in

the solution.

- An other factor is considered in the criterion which is the number of required resources : it might be more difficult to find enough feasible positions for assets with a large number of resource requirements. Hence, it is more interesting to insert them during the early stages.

It is noteworthy to mention that it is more interesting to evaluate these factors at the same scale size. Hence, we propose to divided each factor by its maximum possible value in order to have it within the interval $[0, 1]$.

Let t_{max} be the length of the longest time window, P_{max} be the best profit among all the assets and r_{max} be the maximum number of vehicles required by the assets. The insertion criterion is calculated for each asset as follows :

$$Cr_i(\alpha, \beta, \gamma) = \frac{(\frac{c_i - o_i}{t_{max}})^\beta}{(\frac{p_i}{p_{max}})^\alpha (\frac{r_i}{r_{max}})^\gamma} \tag{4.27}$$

As shown in equation (4.27), the three factors are weighted using the parameters α, β and γ. These parameters are adjusted through the solution process in order to control the relative importance of different factors, and hence, enable the insertion heuristic to cover a large part of the search space. Moreover, several combinations of (α,β,γ) are separately generated at each iteration of ILS in order to boost the convergence of the heuristic. The process used to generate the weights is described as follows : the initial values of α, β and γ are set to 0.5. Then, six functions $f_l, l \in \{1, \ldots, 6\}$ are used to calculate six new combinations of (α,β,γ) at each iteration. In the first four functions $f_l, l \in \{1, \ldots, 4\}$, the value of α is set to 1, whereas the values of β and γ in the previous execution are slightly modified within the interval $[0, 1]$. They are either increased or decreased by a step of 0.1. This results in four different combinations of (β,γ). In the fifth function f_5, β and γ are randomly generated while α is always set to 1. The last function randomly generates all the parameters within $[0, 1]$. At the end of each iteration of the ILS, the combination that led to the solution with the best collected profit is used in the next iteration.

In each iteration of the insertion algorithm, the first asset is chosen from the insertion list. Then, feasible positions in terms of time windows are calculated and one position is randomly selected. The process is reiterated until all the assets in the list are inserted or no feasible insertion is found. Since the feasibility of time windows is checked many times during the search process, an efficient way to verify the feasibility in a constant time is of great importance. To reach that, some information need to be archived and updated after each insertion. We define $Maxshift_i$ as the maximum delay allowed for the service of asset i in case an unrouted asset

get inserted before i. Let us also denote by Arr_i^k the arrival time of vehicle k at asset i, and let Str_i be the service starting time of asset i. Due to synchronization constraints, the starting time service may be delayed so that all the assigned vehicles are present at the asset. It holds that :

$$Str_i = max\{o_i, \max_{k \in R}\{Arr_i^k\}\} \tag{4.28}$$

Following this, if an asset i is visited by vehicle k, the waiting time is :

$$Wait_i^k = Str_i - Arr_i^k \tag{4.29}$$

If vehicle k does not visit the asset i, $Wait_i^k$ is set to $+\infty$. We denote by $k(p)$ the p^{th} asset visited by vehicle k. In order to calculate the $Maxshift_i$ of asset i, we calculate first different $Maxshift_i^k$ for each vehicle $k \in R_i$, where R_i is the set of vehicles that visit asset i :

$$Maxshift_{i=k(p)}^k = min\{c_{k(p)} - Str_{k(p)}, Wait_{k(p+1)}^k + Maxshift_{k(p+1)}\} \quad k \in R_i \tag{4.30}$$

Hence, the value of $Maxshift_i$ is calculated as follows :

$$Maxshift_i = \min_{k \in R_i}\{Maxshift_i^k\} \tag{4.31}$$

On the other hand, if an asset i get inserted in route k between p and $p+1$, the generated shift ($Shift_i^{k,p}$) is calculated as :

$$Shift_i^{k,p} = t_{k(p)i} + Wait_i^k + a_i + t_{ik(p+1)} - t_{k(p)k(p+1)} \tag{4.32}$$

Subsequently, an insertion is feasible if the value of $Shift_i^{k,p}$ is less than or equal to the sum of $Wait_{k(p+1)} + Maxshift_{k(p+1)}$.

Since the routes are interdependent, an update needs to be propagated through all the solutions after each insertion. However, the propagation may loop infinitely if cross synchronizations are not prohibited : when asset j is visited before i in a route, visiting i before j should be prohibited in any other routes. To avoid such situations, another feasibility check called the cross-synchro-feasibility is performed before the insertion of assets that require multiple vehicles. This test is carried out in a constant time ($\mathcal{O}(1)$) thanks to the transitive closures [Aho et al., 1972].

The insertion algorithm is described in Algorithm 8.

Algorithm 8: INSERTION ALGORITHM

Input: *Solution* S, parameters (α,β,γ)

$\sigma \leftarrow$ Sort unrouted assets to non-decreasing values of $Cr_i(\alpha, \beta, \gamma)$
insert = true
while $(insert\ AND\ \sigma \neq \emptyset)$ **do**
  $insert = false$
  for $(i = 0; i < |\sigma|; i++)$ **do**
    $k = 1$
    while $(k \leq r_{\sigma_i})$ **do**
      $Pos \leftarrow$ all positions in S // depends on the type the vehicle
      $foundPos = false$
      foreach $((r, p) \in Pos\ randomly\ selected)$ **do**
        //(r : a route of S, p : position in r)
        if $(k > 1)$ **then**
          if $((r, p)$ *is not cross-synchro-feasible for* $\sigma_i)$ **then** *continue*
        if $((r, p)$ *is not time-window-feasible for* $\sigma_i))$ **then** *continue*
        Insert σ_i in position (r, p)
        Update S
        $foundPos = true$ *break*
      if $(foundPos = false)$ **then** *break*
      $k++$
    if $(k \leq r_i)$ **then** Remove σ_i and update S
    else insert = true
return S

4.5.3 CP-based insertion

The reconstruction phase in based on the CP model, introduced in Section 4.4. The basic idea behind is to use the constraint propagation mechanism of CP in order to further reduce the feasible region of the search space. Given a partial solution S with P is the set of served assets and $\{V_{qp} \,|\, q \in Q, p \in P\}$ is the set of routes, performing the insertion is simply solving CP-STOPTW with the additional following constraints.

The first set of constraints are the *selection constraints*, which are used to impose the presence of the assets that exist in the partial solution S. Let P be the set of served assets in S, the constraints are :

$$PresenceOf(ast_i) = 1 \quad \forall i \in P \tag{4.33}$$

The second family is the *assignment constraints*. it imposes which vehicles are used to visit a given asset i. Assume that asset i is visited by vehicle p of type q, the resulted constraint is :

$$PresenceOf(vst_{ipq}) = 1 \tag{4.34}$$

The third family is the *routing constraints*. It is used to impose the order of visits carried out by a single vehicle.

$$EndBeforeStart(vst_{ipq}, vst_{jpq}) = 1 \quad \forall q \in Q, \forall p \in \{1, \ldots, P_q\}, \forall i \in V^d, j \in V^a : i \xrightarrow{qp} j \tag{4.35}$$

Where constraint $EndBeforeStart(u, v) = 1$ imposes that interval variable u ends before the start of interval variable v, whereas $i \xrightarrow{qp} j$ indicates that asset i is visited before asset j by the vehicle p of type q.

In order to strengthen the model and help the CP solver to find inconsistencies in earlier stages of the search tree, we propose to add the bounds on the time windows calculated in Section 4.5.2. Hence, for each asset $i \in P$,

$$ast_i.StartMin(S.Str_i) \tag{4.36}$$

$$ast_i.StartMax(S.Maxshift_i) \tag{4.37}$$

4.5.4 Post optimization phase

The ILS generates throughout the solving process a set of good solutions. Single tours are then saved in a pool $\mathcal{T}_q = \{T_{1q}, T_{2q}, \ldots, T_{|\mathcal{T}_q|q}\} | q \in Q$. Tour recombination phase consists in solving a modified set cover formulation over $\mathcal{T}_q | q \in Q$ in order to extract a combination of routes that defines the best possible solution for STOPTW. In the following, we propose a mixed integer programming formulation to solve the set cover problem. Let $\theta_{qk},\ 1 < q < |Q|,\ 1 < k < |\mathcal{T}_q|$ be a decision variable which indicates whether route T_{qk} is selected or not. We denote by $i \xrightarrow{qk} j$ two consecutive visits (visit of asset i precedes the visit of asset j) in a given route T_{qk}.

We define also the set matrices $A_q | q \in Q$ as follows :

$$A_q = (a_{iqk}) \ with \ a_{iqk} = \begin{cases} 1 & if \ asset \ i \in k^{th} \ route \ of \ \mathcal{T}_q \\ 0 & otherwise \end{cases}$$

The mathematical formulation, MIP2, is as follows :

$$max \sum_{i=1}^{n} p_i y_i \tag{4.38}$$

$$\sum_{k:T_{qk}\in\mathcal{T}_q} a_{iqk}\theta_{qk} \geq r_{iq}y_i \quad \forall q \in Q, \forall i \in V^- \tag{4.39}$$

$$\sum_{k:T_{qk}\in\mathcal{T}_q} \theta_{qk} \leq P_q \quad \forall q \in Q \tag{4.40}$$

$$s_i+t_{ij}+a_i-s_j \leq M(1-\theta_{qk}) \quad \forall q \in Q, \forall k : T_{qk} \in \mathcal{T}_q, \forall i \in V^d, j \in V^a : i \xrightarrow{qk} j \tag{4.41}$$

$$o_i \leq s_i \leq c_i \quad \forall i \in V \tag{4.42}$$

MIP2 aims at maximizing the total collected profit (4.38) subject to to the set of constraints (4.39-4.42). Constraints (4.39) are the resource requirement constraints whereas (4.40) impose an upper limit on the number of vehicles. Time constraints are initially verified by all the tours present in the pool. However, the combination of different tours can cause a violation of the synchronization constraints. To avoid such issue, we add constraints (4.41) in which only one starting time decision variable s_i is used per asset i. In this way, we impose that a given asset i must be visited by the requested vehicles at the same time.

The pool size is a critical performance parameter. Adding all the feasible tours to the pool may lead to a long computational time. We impose in our case an upper limit on the size of the pool. Furthermore, in order to diversify the tours and to avoid duplicates, we use a hash-based function that determines duplicated solutions and avoids the insertion of their tours in the pool.

We propose also to accelerate the post-optimization phase by computing the upper bound from the LP relaxation of MIP2. The motivation for such approach is two-fold. First, formulations with a large number of variables generally have a tighter relaxation [Barnhart et al., 1998]. Second, since the size of the pool is limited and all the tours are generated by the ILS, the solution returned by MIP2 could be no better than the best solution already found. If it is the case, the solution of MIP2 is skipped and hence, computational efforts are saved for other parts of the algorithm.

4.6 Computational tests

We investigate in this section the performance of the GRASP×ILS method. It was implemented using C++ and STD library, whereas the CP model and the set cover were both solved using the IBM ILOG suite (CP Optimizer 12.6 and IBM ILOG Cplex 12.6 solver) through IBM ILOG Concert Technology. The experimental tests

were conducted on a Linux server running Centos 5.4 and equipped with an Intel Xeon E5420 with 2.66 GHz and 128 GB RAM.

4.6.1 Benchmark instances

Test instances for the STOPTW are generated based on 60 problem instances proposed initially for the VRPTW in [Gehring and Homberger, 1999]. These instances are divided into six main classes : R1, C1, RC1, R2, C2 and RC2 with ten instances under each class. Time windows were modified in the original instances in order to simulate the propagation of the fire fronts across a landscape, whereas the requirements in terms of vehicles was randomly generated and added as a vector of three components, that is, three types of vehicles are considered. Instances in the benchmark are all composed of 200 vertices in addition to the depot, and each instance was used to derive two additional instances by truncating 35 and 100 vertices. When solving these instances, two different configurations of vehicle numbers are used at each size. We obtain then 360 instances, with 120 instances for each size (35, 100 and 200). Test instances are available at http: //www.sites.google.com/site/imanrzbh/datasets.

4.6.2 Performance comparison

In the following, we compare the results of the GRASP×ILS with those of the ALNS proposed in [Roozbeh et al., 2018]. We run our algorithm ten times on each instance and we recorded the average as well as the best result. We consider in a first step small instances (35 assets) with two sets of vehicle numbers ($< 4, 3, 2 >$ and $< 5, 4, 3 >$). Table 4.1 shows a comparison between ALNS and GRASP×ILS with respect to the optimal objective value obtained by CPLEX solver. Results are presented per class and set of vehicle numbers. For each set, we report the average computational time (CPU) in seconds, the gap between optimal solutions and the average objective value (Avg) and finally the gap between the optimal solution and the best run ($Best$).

According to the results depicted in Table 4.1, GRASP×ILS and ALNS have more or less the same computational times with $9.26sec$ for GRASP×ILS and $9.68sec$ for ALNS. Regarding the optimality gaps, our method succeeded to reduce the overall gap between the best result and the optimal solution on all the sets. In fact, GRASP×ILS achieved 0% of optimality gap on all the sets except for sets $R100$ with $< 5, 4, 3 >$ and $R200$ with $< 5, 4, 3 >$. However, these gaps are still less than those of ALNS (0.17 and 0.29 respectively) for the same sets. This good performance of GRAPS×ILS was confirmed by the gap between the average results and the optimal solutions. Here, our method showed a robust behavior with 0.33% of overall gap

against 1.91% for ALNS. Note that the worst overall gap realized by GRASP×ILS was 0.53% for $RC100$ with $< 5, 4, 3 >$ vehicles, whereas ALNS reaches the overall gap of 2.38% for the set $C200$ with $<5, 4, 3>$ vehicles.

Tableau 4.1 – Comparison between ALNS and GRASP×ILS for 35-node instances

Class	#Vehicles	ALNS			GRASP×ILS		
		CPU	Avg (%)	Best(%)	CPU	Avg (%)	Best(%)
$C100$	$<4, 3, 2>$	9.37	1.49	0	8.78	0.34	0
	$<5, 4, 3>$	10.09	2.03	0	9.13	0.31	0
$C200$	$<4, 3, 2>$	9.09	1.49	0	12.11	0.45	0
	$<5, 4, 3>$	9.48	2.38	0	14.23	0.39	0
$R100$	$<4, 3, 2>$	9.84	1.85	0.38	8.46	0.23	0
	$<5, 4, 3>$	10.37	2.31	0.17	9.64	0.23	0.07
$R200$	$<4, 3, 2>$	9.30	2.20	0.23	8.03	0.27	0
	$<5, 4, 3>$	9.42	1.85	0.29	6.63	0.36	0.05
$RC100$	$<4, 3, 2>$	9.44	1.34	0	9.76	0.21	0
	$<5, 4, 3>$	9.9	1.62	0	6.80	0.53	0
$RC200$	$<4, 3, 2>$	9.59	2.10	0.39	9.65	0.42	0
	$<5, 4, 3>$	10.24	2.22	0	7.93	0	0
Mean		9.68	1.91	0.12	9.26	0.33	0.01

Table 4.2 shows a result comparison between GRASP×ILS and ALNS while considering instances with 100 assets. The set of vehicle numbers chosen are proportional to the size of instances in order to cover a considerable portion of assets. The two sets are ($<6, 5, 4>$ and $<7, 6, 5>$). In addition to columns defined in Table 4.1, we added a new column $GAP(\%)$ that displays the average improvement achieved by our method compared to the ALNS. According to results showed in Table 4.2, our method succeeded to achieve substantial improvements compared to ALNS. Computational times were divided by factor 2.9, decreasing from 141.66 seconds for ALNS to only 49.22 seconds. The overall percentage of protected assets was also substantially improved, raising from 68.03% to 76.47%, yielding an overall improvement gap of -12.38%.

Table 4.3 presents the results of both methods while considering large-scale instances with 200 assets. The set of vehicle numbers are ($<9, 8, 7>$ and $<12, 11, 10>$). Results depicted in Table 4.3 confirms the out-performance of the proposed method. Overall computational times achieved by GRASP×ILS are 3.3 times better than

Tableau 4.2 – Comparison between ALNS and GRASP×ILS for 100-node instances

Class	#Vehicles	ALNS			GRASP×ILS			
		CPU	Avg (%)	Best(%)	CPU	Avg (%)	Best(%)	GAP(%)
$C100$	$< 6.5.4 >$	136.12	60.14	61.82	62.82	68.72	70.16	−13.49
	$< 7.6.5 >$	147.59	66.87	68.47	58.41	76.54	77.90	−13.77
$C200$	$< 6.5.4 >$	133.19	59.18	60.83	57.16	65.99	67.08	−10.26
	$< 7.6.5 >$	143.87	65.11	66.88	54.52	73.31	74.58	−11.51
$R100$	$< 6.5.4 >$	135.96	61.92	63.27	45.67	69.16	70.59	−11.56
	$<5, 4, 3>$	139.72	68.80	70.22	44.04	77.55	79.03	−12.54
$R200$	$< 6.5.4 >$	135.75	63.89	65.58	44.38	72.13	73.45	−12.00
	$< 7.6.5 >$	144.16	70.02	71.51	39.80	80.07	81.38	−13.80
$RC100$	$< 6.5.4 >$	144.43	66.49	68.32	48.18	75.34	76.60	−12.11
	$< 7.6.5 >$	149.24	73.21	75.17	45.14	83.05	84.16	−11.96
$RC200$	$< 6.5.4 >$	142.88	67.42	69.32	46.75	76.44	77.79	−12.21
	$< 7.6.5 >$	146.97	73.33	74.97	43.82	83.86	84.97	−13.34
Mean		141.66	66.37	68.03	49.22	75.18	76.47	−12.38

that of ALNS, with 176.36 seconds against 578.93 seconds. The overall percentage of saved assets was also improved which attains 74.40% against 64.04% realized by ALNS, yielding an overall improvement gap of −16.26%.

4.7 Conclusion and perspectives

In this chapter, we were interested to a new variant of the Team Orienteering Problem called the Synchronized Team Orienteering Problem with Time Windows. This problem was originally proposed in order to model and solve asset protection problem during escaped wildfires. To solve this problem, we proposed a metaheuristic scheme of type GRASP×ILS that incorporates a CP-based insertion module. A post optimization phase is also added to the GRASP×ILS in order to improve the best solution. Computational results proves the efficiency of our approach when compared to the literature. As future work, we will consider an additional criterion in the objective function which is the load balancing. This criterion is relevant when we intend to equally distribute activities on protection teams and avoid them to be overwhelmed. Another research direction is the elaboration of an exact method

Tableau 4.3 – Comparison between ALNS and GRASP×ILS for 200-node instances

Class	#Vehicles	ALNS			GRASP×ILS			
		CPU	Avg (%)	Best(%)	CPU	Avg (%)	Best(%)	GAP(%)
$C100$	$< 9.8.7 >$	589.60	56.13	57.68	199.06	65.41	66.37	−15.08
	$< 12.11.10 >$	619.33	65.11	66.57	213.02	76.88	77.65	−16.64
$C200$	$< 9.8.7 >$	542.64	51.06	52.60	187.57	60.81	62.07	−18.00
	$< 12.11.10 >$	566.36	60.34	61.46	209.98	72.67	73.66	−19.85
$R100$	$< 9.8.7 >$	539.19	58.23	59.60	134.92	67.83	68.92	−15.63
	$< 12.11.10 >$	585.49	69.04	70.29	163.12	81.13	82.26	−17.03
$R200$	$< 9.8.7 >$	542.78	57.75	59.27	148.60	67.76	68.94	−16.32
	$< 12.11.10 >$	589.75	68.74	70.17	170.68	80.78	81.84	−16.63
$RC100$	$< 9.8.7 >$	561.80	60.90	62.18	158.75	71.38	72.30	−16.29
	$< 12.11.10 >$	607.04	71.21	72.46	185.83	84.23	85.19	−17.56
$RC200$	$< 9.8.7 >$	570.06	61.45	62.66	157.70	71.95	73.14	−16.72
	$< 12.11.10 >$	633.17	72.14	73.58	187.14	84.78	80.44	−9.32
	Mean	578.93	62.68	64.04	176.36	73.80	74.40	−16.26

based on the new formulation proposed in this chapter.

Memetic Algorithm with a dynamic programming-based splitting procedure for the Set Orienteering Problem

Sommaire

5.1 Introduction

The Traveling Salesman Problem (TSP) is a well-studied combinatorial optimization problem. It aims to find a circuit visiting all customers from a central depot while minimizing the travel distance. TSP with profit (TSPP) is a variant of TSP in which each customer is associated with a profit to represent the value of service and this profit is gained only if the customer is visited. This characteristic leads to the definition of three main categories of TSPPs [Archetti et al., 2014] : the Orienteering Problem (OP), where the objective is to maximize the collected profit with respect to a limited travel time. The Prize Collecting TSP (PCTSP) which aims to minimize the travel cost while guaranteeing a minimum collected profit. The Profitable Tour Problem (PTP) where the goal is to maximize the difference between collected profit and costs.

Among these three categories, the OP, also known as the Selective Traveling Salesman Problem, is the most studied problem in the literature [Angelelli et al., 2014]. This problem is inspired from the sport game of orienteering as described in [Chao et al., 1996]. Many exact and heuristic methods have been proposed for the problem. The reader can refer to [Feillet et al., 2005],[Archetti et al., 2014] and [Vansteenwegen et al., 2011] for excellent surveys on variants, applications and also resolution methods.

Recently, a new variant of the OP was proposed in [Archetti et al., 2018] called the Set Orienteering Problem (SOP). In this variant, the set of customers are grouped into subsets called *clusters*. A profit is assigned to each cluster and it is gained if at least one of its customers is visited. A single vehicle with a limited travel time is available to serve the clusters. The objective is to select a subset of clusters to serve in a way that the total collected profit is maximized with respect to the travel time constraint.

Some relevant applications for the SOP can be found in the distribution of mass products. For example, consider the case of a carrier that signs contracts with different supply chains, each one composed of a set of retailers. In order to reduce transportation costs, supply chains may stipulate in the contract that it is not necessary to serve all its retailers. Instead, the carrier had to deliver the entire quantity requested by the chain to only one of its retailers, while the inner distribution is performed by the internal logistic network.

5.2 Basic mathematical model

We consider a complete directed graph $G = (V, A)$ where $V = \{1, \ldots, n\} \cup \{0, n+1\}$ is the vertex set representing the customers and the depots. For convenience, we use the sets V^d, V^a and V^- to denote respectively the vertex set without the arrival depot, the vertex set without the departure depot as well as the vertex set without depots. The arc set $A = \{(i, j) | i, j \in V, i \neq j\}$ represents the connections between customers and each arc $(i, j) \in A$ is associated with the travel time t_{ij}. The travel times are assumed to be asymmetric and satisfy triangular inequality. A partition $S = \{S_1, S_2, \ldots, S_K\}$ of V is given where a node subset S_l is called a *cluster*. A profit P_l is associated with each cluster S_l and it is gained if at least one of its customers is visited. A single vehicle with a limited travel time T_{max} is available to serve the customers.

Before introducing the mathematical model, let us first present some useful notation. For each set $U \subseteq V^-$, let

- $A(U) = \{(i,j) \in A | i \in U, j \in V \in U\}$,
- $\delta^+(U) = \{(i,j) \in A | i \in U, j \in V^a \backslash U\}$,
- $\delta^-(U) = \{(i,j) \in A | i \in V^d \backslash U, j \in U\}$.

For the ease of notation, if $U = \{j\}$, we simply write $\delta^+(j)$ or $\delta^-(j)$ instead of $\delta^+(\{j\})$ or $\delta^-(\{j\})$.

Decision variables used in the model are :

- z_l : equal to 1 if cluster $S_l \in S$ is served, 0 otherwise,
- y_i : equal to 1 if customer $i \in V$ is visited, 0 otherwise,
- x_{ij} : equal to 1 if arc $(i,j) \in A$ is traversed by the vehicle, 0 otherwise.

The mathematical formulation for the SOP, $ILP1$, is as follows :

$$max \sum_{l:S_l \in S} z_l P_l \tag{5.1}$$

$$\sum_{(h,i) \in \delta^-(i)} x_{hi} = y_i \quad \forall i \in V^a \tag{5.2}$$

$$\sum_{(i,j) \in \delta^+(i)} x_{ij} = y_i \quad i \in V^d \tag{5.3}$$

$$\sum_{(i,j) \in E(U)} x_{ij} \leq |U| - 1 \quad \forall U \subseteq V^- \tag{5.4}$$

$$\sum_{(i,j) \in A} t_{ij} x_{ij} \leq T_{max} \tag{5.5}$$

$$z_l \leq \sum_{i \in S_l} y_i \quad \forall S_l \in S \tag{5.6}$$

$$z_l \in \{0,1\} \quad \forall S_l \in S \tag{5.7}$$

$$y_i \in \{0,1\} \quad \forall i \in V \tag{5.8}$$

$$x_{ij} \in A \quad \forall (i,j) \in A \tag{5.9}$$

The objective function (5.1) aims at maximizing the total collected profit. Constraints (5.2) and (5.3) are flow conservation constraints. Subtours elimination is

guaranteed by constraints (5.4). Constraint (5.5) imposes a travel time limit on the vehicle. Constraints (5.6) impose that the vehicle must visit at least one customer per cluster. Constraints (5.7)-(5.9) are domain definitions.

Clearly, this formulation has an exponential number of subtour elimination constraints (5.4). In practice, these constraints are initially removed from the model and inserted dynamically once violated during the solution process. The process of identification of violated SECs in a branch-and-cut method is commonly referred to as separation procedure. Several procedures have been proposed to separate the SECs. We present in this section three methods. The first one is based on the well-know maximum flow/minimum-cut algorithm applied usually on fractional solutions inside the branch-and-bound tree. The second approach is based on the computation of the strong components of the support digraph of integral solutions. In the third approach, we propose to combine the two previous separation procedures. These methods are detailed in the following.

5.2.1 Branch-and-cut algorithm

Subtour elimination constraints are replaced by the well-known MTZ formulation which has a polynomial number of constraints. To that end, an additional decision variable u_i is considered. The MTZ constraints are :

$$u_0 = 1 \tag{5.10}$$

$$u_i - u_j + 1 \leq (n-1)(1 - x_{ij}) \quad \forall i, j \in V^- \tag{5.11}$$

$$2 \leq u_i \leq n \quad \forall i \in V^- \tag{5.12}$$

Clearly, MTZ constraints guarantee the elimination of subtours. To further strengthen the formulation, we propose to add another type of SECs called the Generalized Subtour Elimination Constraints (GSECs).

$$\sum_{(i,j)\in\delta^+(U)} x_{ij} \geq y_v \quad \forall U \in V^-, v \in U \tag{5.13}$$

Of course, as the number of the GSECs is exponential, they are dynamically added to the model once violated. The separation of GSECs is applied on fractional solutions in the branch-and-bound tree using the following exact procedure.

Let (x^*, y^*, z^*) be an optimal fractional solution in a given node of the branch-and-bound tree. Let $G^* = (V, A^*)$ be a directed graph where we consider only

arcs $(i,j) \in A$ for which the corresponding x^*_{ij} has a non-zero value. Each arc in $(i,j) \in A^*$ is associated with a capacity equal to the value of x^*_{ij}. For each vertices $i \in V$, we solve the max-flow/min-cut from i as a source to $n+1$ as a sink on the graph G^*. Vertices in G^* are considered in a non-increasing order of the associated y^*_i. If the min-cut induces a partition U where $i \in U$ and $n+1 \notin U$ and the value of the maximum flow is less than y^*_i, then a violated GSEC is identified. After the separation of a GSEC, an additional capacity equal to $1 - \sum_{(i,j)\in\delta^+(i)} x^*_{ij}$ is added to $x^*_{i,n+1}$ in order to prevent the identification of the same cut in subsequent iterations. In our procedure, the max-flow/min-cut problem is solved using the algorithm proposed in [Boykov and Kolmogorov, 2004]. The complexity of this algorithm is $\mathcal{O}(|C||A^*|V^2)$ where $|C|$ is the min-cut. Trying all possible vertices in V yields to an overall complexity of $\mathcal{O}(|C||A^*||V|^3)$.

5.2.2 Cutting planes algorithm

In the cutting planes approach, constraints (5.4) are relaxed from the basic formulation. The new compact model is then solved to integer optimality. Clearly, the resulted solution may contain subtours. If it is the case, violated SECs are added to the model and the process is repeated until no subtour is found. The identification of subtour in this case can be easily done by computing strong components in a directed graph. In fact, a subtour in an integral solution is equivalent to a sub-diagraph with one non-trivial strong component in the support digraph. Strong components can be calculated using Tarjan algorithm introduced in [Tarjan, 1972] which is based on the Depth First Search algorithm with time complexity of $\mathcal{O}(|V|^2)$. Obviously, the efficiency of this approach is tightly related to the performance of ILP-solvers, which have been considerably improved during the last decades.

In the purpose of eliminating the identified subtours, we use the Generalized SECs introduced in [Fischetti et al., 1998].

$$\sum_{(i,j)\in\delta^+(U)} x_{ij} \geq y_v \quad \forall U \in V, \{0, n+1\} \subseteq U, v \in V\backslash U \tag{5.14}$$

$$\sum_{(i,j)\in A(U)} x_{ij} \leq \sum_{u\in U\backslash\{0,n+1\}} y_u - y_v + 1 \quad \forall U \in V, \{0, n+1\} \subseteq U, v \in V\backslash U \tag{5.15}$$

$$\sum_{(i,j)\in A(U)} x_{ij} \leq \sum_{u\in U\backslash\{0,n+1\}} y_u - y_v \quad \forall U \in V^-, v \in U \tag{5.16}$$

5.2.3 Enhanced branch-and-cut algorithm

In this approach, we combine between the two separation procedures. We use max-flow/min-cut to separate SECs in fractional solutions, while we use Tarjan algorithm to identify SECs in integral solutions. That is, instead of adding MTZ constraints to the basic model as in Section 5.2.1, we solve the model with MTZ and once an integral solution is found in a given node of the branch-and-bound tree, we verify if there are any strong components. If it is the case, the corresponding GSECs (5.14), (5.15) and (5.16) are added to the model and ILP-solver pursues then the branching process.

5.3 Memetic Algorithm

Memetic Algorithm (MA) is a combination of Genetic Algorithm (GA) with local search techniques. The GA is a population-based metaheuristic that explores the search space using a population of individuals or solutions. This population evolves throughout the search process as new individuals are generated using a crossover operator. The latter picks up two or more individuals from the population and combines them to produce a new one. In MA, a local search method is used to improve the new generated individuals. In other words, MA uses the GA crossover operator to explore promising regions of the search space, whereas the local search is used to concentrate the search around these regions. In MA/GA, solutions are encoded into a uniform structure called *chromosome*. A chromosome is generally associated with a unique solution, but it can also represent a neighborhood of solutions. In this case, the chromosome can be seen as an *indirect representation* of solutions which needs a decoding procedure in order to retrieve a feasible solution.

In our method, we use an indirect representation called the *giant tour* which is a tour that covers all the customers. It represents a neighborhood of solutions with a common characteristic : the visiting order of customers should be the same as the giant tour. The extraction of a solution (decoding) is performed by an optimal splitting procedure that extracts the best feasible solution in the neighborhood of the giant tour.

The splitting procedure is a core component of memetic algorithms proposed to solve VRPs. The first splitting procedure was proposed by Beasley in [Beasley, 1983] for the Capacitated Vehicle Routing Problem (CVRP). It was then incorporated by Prins [Prins, 2004] into a complete framework (a genetic algorithm) to solve the general CVRP. Since then, numerous splitting procedures have been proposed for different variants of VRP and they showed an outstanding performance compared

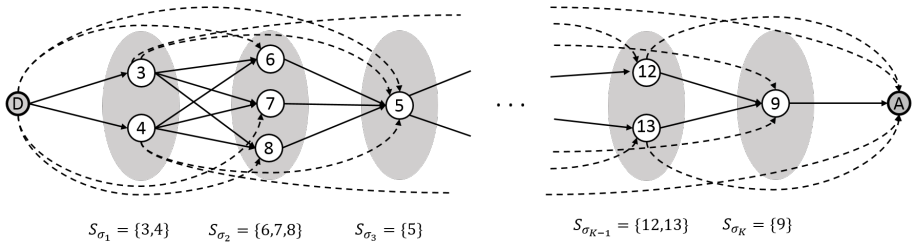


Figure 5.1 – Auxiliary graph representing some of the possible arcs

to other state-of-the-art approaches. We propose in this section a new splitting procedure specific for the SOP based on dynamic programming. This method explores a neighborhood covering an exponential number of solutions in a pseudo-polynomial time and space complexity.

5.3.1 Splitting procedure

A giant tour in our case is a sequence of nodes $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \ldots, \sigma_K, \sigma_{|\sigma|}\}$ where each node σ_j represents the cluster $S_{\sigma_j} \in S | 1 < \sigma_j < K$ whereas σ_0 and $\sigma_{|\sigma|}$ are the departure and the arrival depots. A node $\sigma_j | 1 \leq j \leq K$ contains a set of sub-nodes that represent the customers belonging to the associated cluster. The aim of the splitting procedure is to select at most one customer to visit per cluster in order to maximize the total collected profit while respecting the original routing order of clusters and the time constraint. This sub-problem can be modeled by a shortest path problem with resource constraints on a directed acyclic graph $H = (V, A_\sigma)$ where $A_\sigma = \{(\sigma(i), \sigma(j)) \in A | i < j\}$ as illustrated in Fig. 5.1.

Figure 5.1 depicts a giant tour σ composed of K clusters in addition to the departure and the arrival depots, each cluster is represented by a grey ellipsoidal node within which are represented its customers. Continued arrows represent arcs between customers belonging to consecutive clusters in σ, whereas discontinued arrows represent arcs that skip at least one cluster in σ. Note that only a subset of arcs are represented in order not to overload the illustration.

We solve the shortest path problem with resource constraints using dynamic programming and label propagation technique. A label is used to represent a path from the departure to a given node σ_i. Each label records all the visited customers as well as the collected profit and the travel time.

We define a label L by the following attributes :

- L^p : the profit of clusters covered in the path

- L^t : the travel time of the path

For performance reasons, instead of storing all the labels in one queue, each customer $u \in S_{\sigma(i)}$ maintains a separate queue $\mathcal{Q}_u^i$. Note that the departure node is initialized by a single queue with a single label $\mathcal{Q}_0^0 = \{(0,0)\}$. The extension of a label $L \in \mathcal{Q}_u^i$ to any other customer v in a successor node $\sigma_{i+k}|1 \leq k$ yields to the creation of a new label $R \in \mathcal{Q}_v^{i+k}$ with $R^p = L^p + p_{\sigma_i}$ and $R^t = L^t + c_{u,v}$. Indeed, before creating the new label, the travel time constraint should be respected, that is $L^t + c_{u,v} + c_{v,n+1} \leq T_{max}$. Otherwise, the label is fathomed.

In order to reduce the number of labels, a dominance property is used to detect and remove inferior labels. Given two labels $L = (L^p, L^t)$ and $R = (R^p, R^t)$ stored at the same queue $\mathcal{Q}_u^i$, L dominates R if and only if : $L^p > R^p$ and $L^t \leq R^t$.

5.3.1.1 Knapsack-based upper bound

In order to reduce the number of labels and accelerate the splitting procedure, we propose to calculate an upper bound based on the Knapsack Problem (KP). This bound helps to detect and fathom unpromising labels and hence, limit their proliferation in earlier stages of the procedure.

Let σ_i be the last node visited by label L. Also, let $S' = \{S_{\sigma_{i+1}}, \ldots, S_{\sigma_K}\}$ be the set of clusters associated with the remained nodes that lie after the node σ_i in σ. The KP model can be described as follows. Each cluster $S_j \in S'$ is represented by an item j in the KP model and it is associated with a profit p_j. The weight w_j of item j is modeled by the minimum cost insertion among all the possible insertions of the customers of its associated cluster S_{σ_j}. w_j is calculated as :

$$w_j = \min_{u \in S_{\sigma_j}} \{\frac{1}{2} \min_{\substack{v \in S_{\sigma_k} \\ 0<k<j-1}} \{c_{vu}\} + \frac{1}{2} \min_{\substack{v \in S_{\sigma_k} \\ j+1<k<|\sigma|}} \{c_{uv}\}\} \tag{5.17}$$

Let W_u^i be an upper bound on the remaining travel time incurred to cover the clusters of S'. Assume that $L \in \mathcal{Q}_u^i$, W_u^i is calculated as :

$$W_u^i = T_{max} - (L^t + \frac{1}{2} \min_{\substack{v \in S_{\sigma_k} \\ i+1 \leq k \leq |\sigma|}} \{c_{u,v}\} + \frac{1}{2} \min_{\substack{v \in S_{\sigma_k} \\ 0 \leq k < i}} \{c_{v,n+1}\}) \tag{5.18}$$

According to the KP model, it is clear that all the weights are underestimated, whereas the knapsack size is overestimated. Hence, solving this model yields to an upper bound γ on the profit that can be collected by any sub-path extended from L. Given now the best-found profit P_{best}, the label L is fathomed **iff** $L^p + \gamma \leq P_{best}$.

It is clear that solving a large number of small KP models can dramatically increase the computational burden of the splitting procedure. To deal with this

drawback, we propose to calculate at the beginning and once for all a larger KP model using dynamic programming procedure and use intermediate results as solutions for the smaller KP models. The key idea behind is to solve the KP in a backward manner. Let us first denote by $\Gamma(j, w)$ the maximum profit gained by solving a KP on the nodes of the sub-sequence $\phi = \{\sigma_j, \ldots, \sigma_K\}$ with a knapsack size equal to w. The largest KP model is solved by computing iteratively $\Gamma(j, w)|1 < j \leq K + 1, 0 \leq w \leq T_{max}$ in a recursive fashion using the following equation :

$$\Gamma(j-1, w) = \begin{cases} \Gamma(j, w) \quad if \quad w < w_j \\ \max\{\Gamma(j, w), \Gamma(j, w - w_j) + p_{j-1}\} \quad otherwise \end{cases} \tag{5.19}$$

where $\Gamma(K + 1, w) = 0$ for a dummy node $K + 1$ and for all $0 \leq w \leq T_{max}$. As a result, in order to know whether a label $L \in \mathcal{Q}_u^j$ should be fathomed, we only need to verify in $\mathcal{O}(1)$ if $L^p + \Gamma(j + 1, W_u^j) \leq P_{best}$.

5.3.1.2 Quick evaluation

Computational complexity of the splitting procedure is $\mathcal{O}(n^2 D)$, where D is an upper bound on the number of labels per sub-node. Applying such method each time could have a substantial impact on computational times. We propose a faster version with a reduced complexity that we can use in neighborhood search operators that need a considerable number of evaluations.

The quick evaluation is based on the assumption that a path cannot skip more than J nodes in the giant tour, i.e. any arc $(i, j) \in A_\sigma$ is systematically fathomed if $i + J < j$. As a result, the complexity of the quick evaluation becomes $\mathcal{O}(nJD)$.

5.3.2 Algorithm initialization

Random initialization of the population is crucial for the MA to avoid premature convergence and allows to explore large parts of the search space. However, it is always interesting to incorporate some good individuals to guide the global search process and reduce the algorithm time. To that end, we propose to use an Iterative Construction/Destruction (IDCH) heuristic to generate a small part of the initial population.

The main components of IDCH is the Best Insertion Algorithm (BIA) and the Destruction Algorithm (DA). The BIA tries to construct feasible solutions by inserting customers in their best positions according to a greedy criterion. On the other hand, the DA disturbs the constructed solution by removing a subset of the inserted customers.

The insertion of customers in the current solution is carried out iteratively. At each iteration, BIA calculates the best position with the minimal cost insertion for each customer j. The insertion of customer j between two successive customers u and v incurs an insertion cost $\Delta^j_{u,v} = c_{uj} + c_{jv} - c_{uv}$. Possible insertions are then ordered according to the sorting criterion $\frac{(\Delta^{u,v}_j)^\alpha}{P_i}, i \in S_j$, where α is a randomly generated value at each iteration using a uniform distribution in $[0.5, 1]$. In case of ex-aequo, one insertion is chosen at random. Note that once a cluster has an inserted customer, the remainder set of its customers are no longer considered in the following iterations. The insertion process is repeated until all the clusters are served or no feasible insertion is found.

After the construction of an initial solution from scratch using BIA, IDCH applies the DA to disturb the current solution. A number of customers are randomly selected and removed. This number is randomly chosen between 1 and a diversification parameter denoted by d_{max}. After the partial destruction, the neighborhood search operator 2-opt is used to reduce the total travel time of the solution. A repair phase is then performed by applying the BIA and then a new phase of destruction/construction is launched again. In the same time, IDCH records the best solution found so far and updates it after each iteration. In case where this best solution is not improved after K^2 iterations, the search process is terminated. Note that the diversification parameter d_{max} is crucial for the performance of IDCH : if it is too small, the heuristic may not be able to escape from local optima, whereas if it is too large, IDCH would restart each time the construction from scratch. Hence, we proposed an adaptive mechanism to update d_{max}. It is first initialized at 3, then increased by one after each iteration without improvement. Once a new best solution is found, d_{max} is reset to 3.

5.3.3 Local search

To enhance the genetic algorithm, we integrate a local search engine inside to apply mutations on *giant tours* after the crossover operator with *pm* probability. The LS is composed of three operators selected in a random order. In each iteration the algorithm selects an operator randomly which is stopped as soon as an improvement is found. This process ends when all the operators fail to improve the current solution. In the following a brief description of the operators :

- **Shift operator :** Randomly selected customer is extracted from the giant tour and inserted in all other positions.
- **Swap operator :** Randomly selected couple of customers are exchanged.

The evaluation of the giant tour is called in each iteration of the previous operators. In order to reduce the impact on execution time, we use the fast version of the evaluation.

- **Destruction/repair operator :** After the extraction of a solution from a given giant tour, the operator randomly selects at most γ customers and remove them from the solution. Then, clusters are inserted randomly one by one as much as possible using the BIA. Finally, a giant tour is reconstructed from the solution.

5.3.4 General flow

We have chosen the MA as an outer metaheuristic to guide the search process. In other words, the MA generates giant tours and provide them to the splitting procedure for evaluation. The algorithm starts with an initial population of giant tours. At each iteration, two giant tours are selected by binary tournament, on which a crossover operator called Linear Order Crossover LOX is applied to generate a new individual. The latter is evaluated using the dynamic programming based splitting procedure. If there is a giant tour with the same profit and the same length, it is replaced by the new child. Otherwise, the child is inserted in the population and the worst individual in the new pool is removed.

Algorithm 9 summarizes the MA.

5.4 Preliminary results

In this section, we present preliminary computational tests carried out to evaluate the performance of our methods. The implementation was done in C++ using the STD library. Both algorithms were tested on a Linux server running Centos 7 and equipped with an Intel Xeon Gold 6138 @ 2.00 GHz and 180 GB RAM. The mathematical model was implemented using IBM ILOG CPLEX 12.6 solver through Concert Technology.

In the following, we present benchmark instances of the SOP, then we provide preliminary results for the proposed methods.

5.4.1 Test instances

Benchmark instances of the SOP [Archetti et al., 2018] were derived from those proposed for the Generalized Traveling Salesman Problem (GTSP) [Fischetti et al., 1997]. Fifty one class of instances were generated from 51 GTSP

Algorithm 9: MEMETIC ALGORITHM

```
Input: Solution POP
Output: Solution X_best
Initialize population with N giant tour;
repeat
    select 2 parents P_1 and P_2 using binary tournament;
    Gt ← LOX(POP[P_1],POP[P_2]);
    if rand(0,1) < pm then
        LS(Gt);
    if fit(Gt) ≥ fit(POP[0]) then
        if ∃p||fit(POP[p])==fit(Gt) then
            eject POP[p] from POP;
            update stopping condition;
        else
            eject POP[0] from POP;
            reset stopping condition;
        insert or replace Gt in right place in POP;
    else
        update stopping condition;
until (stop condition);
return X_best
```

instances with a number of vertices ranging from 51 to 1084, whereas the number of clusters of each instance is equal to one fifth the number of customers. As the customers are already grouped into clusters in GTSP instances, it remains the assignment of a profit to each cluster and the determination of a suitable T_{max}. Authors in [Archetti et al., 2018] proposed two patterns to generated profits : in the first pattern $p1$, each cluster was assigned a profit equal to the number of its customers. Profits in the second pattern $p2$ are generated in a pseudo-random way where each customer j is assigned a profit equal to $(7141j)mod(100)$. The profit of a given cluster is calculated by summing up the profits of its customers. Regarding T_{max}, three values were considered. Let $GTSP^*$ be the GTSP solution value of a given instance, T_{max} is calculated as $\theta \times GTSP^*$ where θ takes the following values : $\frac{2}{5}$, $\frac{3}{5}$ and $\frac{4}{5}$. In total, 306 SOP instances are generated. In addition, another set of instances is generated through the same process described above, but on the basis of modified instances of the GTSP. The difference consists in the partition of the customers on different clusters which is performed in a random manner. Thus, the total number of instances generated for the SOP is 612.

Preliminary tests were conducted only on a subset of instances with up to 200 vertices and 40 clusters, that is 348 instances.

5.4.2 Computational resutls for the exact method

Table 5.1 depicts the overall performance of each method. The first column reports the name of the separation methods where Mf/Mc stands for the max-flow/min-cut method, SCC stands for the strong connected components method and MTZ for the MTZ subtour elimination constraints. The second column GAP provide the gap between the lower and the upper bound achieved by each method and it is calculated as $\frac{UB-LB}{UB}$. The two last column CPU and $\#OPT$ reports respectively the average computational times and the number of instances solved to optimality by each method. From Table 5.1, it is easy to notice that the combination of the two separation schemes succeeds to obtain the best results. the "Mf/Mc and SCC" approach achieves an average computational time of 2260.42s against 2461.89s for the "Mf/Mc and MTZ" and 2746.86s for "SCC". Similarly, the number of solved instances by "Mf/Mc and SCC" is equal to 156 against 140 achieved by "Mf/Mc and MTZ" and only 99 instances by "SCC". However, "Mf/Mc and SCC" has a major drawback which is the poorness of its feasible solutions in case where it fails to find the optimal solution. This drawback leads obviously to a larger overall gap compared to "Mf/Mc and MTZ", with 41.56% for "Mf/Mc and SCC" against 31.68% for "Mf/Mc and MTZ".

Tableau 5.1 – Comparison between separation procedures

	GAP (%)	CPU(s)	#OPT
Mf/Mc and MTZ	31.68	2461.89	144
SCC	58.68	2746.86	99
Mf/Mc and SCC	41.56	2260.42	156

5.4.3 Preliminary results of the Memetic Algorithm

Authors in [Archetti et al., 2018] proposed a mathheuristic approach to solve the SOP called the MASOP. We provide in this section some preliminary results of our method in comparison with the MASOP.

Table 5.2 shows some performance measures of the heuristic methods. Instances from both sets are considered where each set is divided into three subsets according to the value of θ. We report for the MASOP the average computational times "CPU" in seconds, the gap to the best upper bound found by the exact methods "GAP UB" and the number of instances for which the MASOP found the best solution "#Best".

Tableau 5.2 – Comparison between Memetic Algorithm and MASOP

Class		MASOP			Memetic Algorithm			
SET	θ	CPU(s)	GAP UB (%)	#Best	CPU(s)	GAP UB (%)	GAP MASOP (%)	#Best
1	T40	7.57	12.97	4	1.46	12.87	−0.15	4
1	T60	8.95	15.09	1	17.46	15.08	−0.02	4
1	T80	8.41	11.32	3	39.00	11.25	−0.09	6
2	T40	6.79	3.99	6	7.66	3.99	−0.46	11
2	T60	13.70	1.63	11	8.58	1.66	0.02	4
2	T80	14.68	0.02	0	2.94	0.02	0.00	0
Mean		10.02	7.50	25	12.85	7.48	−0.12	29

Regarding the Memetic algorithm, in addition to "CPU", "GAP UB" and "#Best", we report also the gap to the best results obtained by the MASOP, the number of instances for which the Memetic algorithm found the best solution.

From Table 5.2, we note that the MASOP and the Memetic Algorithm have approximately the same computational times with less that $10.02s$ for MASOP against $12.85s$ for the Memetic algorithm. The observation can be made when considering the gap to the best upper bound, as they have approximately the same value. However, the Memetic Algorithm succeeds to achieve an overall gap of -0.12% compared to the best solutions obtained by the MASOP. Finally, our method improved the best solution for 29 instances of the literature, whereas it failed to find the best solution for 25 instances.

5.5 Conclusion

In this chapter, we tackled a new variant of the OP called the Set Orienteering Problem (SOP). In a first step, we investigated several separation procedures used to identify violated subtour elimination constraints. Tests carried out on small instances showed that the combination between the max-flow/min-cut separation procedure used in fractional nodes in one hand, and in the other hand, the strong connected component based separation procedure used in integral nodes, yields to best results. In a the second part, we proposed a Memetic algorithm (MA) to solve the SOP. The MA incorporates an efficient splitting procedure based on dynamic programming and enhanced by a knapsack-based upper bound. In preliminary tests, the MA improved the best solution for some instances and succeeded to achieve a good gap to the

best results. Further parameter tuning experiments will be carried out in order to achieve the best performance of out method.

Conclusion and future work

In this dissertation, we presented a number of solution methods to solve some variants of the Team Orienteering Problem. The studied problems are widely used in transportation systems in order to maximize the gain while minimizing the total costs. The proposed methods include a number of mathematical models, exact methods as well as heuristics. To prove the performance of the proposed methods, computational tests and comparison with the best results in the literature were provided.

After introducing some useful definitions related to combinatorial optimization in Chapter 1, we introduced in Chapter 2 a new variant of the Team Orienteering Problem that we called the Clustered Team Orienteering Problem (CluTOP). We first introduced the mathematical formulation of the problem. In order to solve the CluTOP, we proposed an exact method based on the cutting plane 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 reduce the number of variables in the mathematical model. Pre-processing techniques include the identification of inaccessible components (customers or clusters), mandatory clusters and incompatibilities between customers and clusters. In addition, valid inequalities like symmetry breaking cuts, bounds on the number of clusters and the collected profit are also considered. The cutting plane algorithm aims at solving a mathematical formulation of CluTOP with a polynomial number of variables and constraints. Initially, subtour elimination constraints are removed from the model, then added progressively once they are not respected. Experimental results confirmed the performance of our approach. A large number of instances were solved to optimality still unsolved by the method in the literature. Moreover, the upper bound of several instances were substantially improved.

In Chapter 3, we were interested in solving the CluTOP using a heuristic method. Our method is based on the *order first-cluster second* approach. We proposed an efficient splitting procedure in order to extract an improved solution from a given giant tour. This method is based on a branch-and-bound scheme that incorporates a feasibility check to fathom unfeasible nodes and a knapsack-based upper bound

to discard unpromising nodes. The splitting procedure was incorporated in a global scheme alongside an Adaptive Large Neighborhood Search heuristic used to generate giant tours of a good quality. According to experimental results conducted on benchmark instances from the literature, our approach outperforms the existing methods where the solution of many instances were improved in less computational time.

The second problem treated in this thesis is the Synchronized Team Orienteering Problem with Time Windows (STOPTW). This variant was recently proposed to model situations of asset protection during escaped wildfire. we presented in Chapter 4 a new mathematical formulation for the STOPTW along with a Constraint Programming based model. We proposed then a metaheuristic approach based on Greedy Randomized Adaptive Search Procedure with Iterated Local Search - GRASP couple with an Iterated Local Search - ILS. This method incorporates two insertion methods : the first one is based on an adaptive candidate list-based insertion procedure that inserts customers iteratively in the incumbent solution. The ordering of customers in the list is continuously modified during the search progress in order to cover a larger space. Our method incorporates also a CP-based insertion procedure that tries to insert optimally the unrouted customers in the incumbent solution. A post optimization phase is performed using a set cover formulation to improve the results. Experimental results showed that our approach dominates the literature by improving the best results of all most all the benchmark instances with a less computational time.

Finally, in Chapter 5, we tackle a recent variant of the Orienteering Problem called the Set Orienteering Problem (SOP). After presenting the mathematical model with an exponential number of constraints, we investigated the different approaches to separate subtour elimination constraints in the purpose of exactly solving the mathematical formulation. Among the three presented approaches, we deduced that the best approach is to combine the two methods of separation, i.e. the max-flow/min-cut method in fractional nodes and the strong connected components method in integral nodes. In addition to the exact method, we proposed as a Memetic Algorithm approach to solve the SOP. An optimal splitting procedure was also proposed in order to extract the best solution from a given giant tour. This procedure is based on a dynamic programming algorithm with label propagation. We proposed also a knapsack-based upper bound in order to prune inferior labels an limit their proliferation. Only a preliminary results were presented for the Memetic algorithm. We intend to carry out an extensive computational tests in order to fine tune different parameters of the algorithm.

Future work

In the following, we outline some directions for further research based on the observations made in this thesis.

A first relevant direction is the consolidation of the work carried out on the Synchronized Team Orienteering Problem with Time Windows. Extended computational tests and sensitivity analyses are planned in order to investigate the contribution of different components in the search process. Another ongoing direction consists in the elaboration of an exact method to solve STOPTW based on the new formulation proposed in Chapter 4. An efficient pre-processing procedure is in construction in order to reduce the search space of the problems. Several interesting components are embedded, mainly incompatibility-cuts based on cliques and energetic reasoning. In addition, further cuts from the literature are planned to be embedded in the framework. This includes Comb Inequalities, Box Inequalities, etc. [Bard et al., 2002].

Regarding the Set Orienteering Problem, extensive computational tests will be carried out to fine tune the proposed algorithm. We will further integrate additional components to our method such as local search operators. We face also to continue improving the branch-and-cut method for the SOP. Besides the separation procedures for subour elimination constraints, we plan to integrate different inequalities that have been proposed for the OP and its variants, like incompatibility cuts, Comb Inequalities, Box Inequalities, Matching constraints, Cover inequalities and Path Inequalities [Fischetti et al., 1998], [Fischetti et al., 1997].

We intend also to extend our research to cover new variants of the TOP as the Capacitated Team Orienteering Problem, the Split Delivery Capacitated Team Orienteering Problem and the Incomplete Service and Split Deliveries in a Routing Problem with Profits. We are also interested in a new class of vehicle routing problems called Rich Vehicle Routing Problems. Problems of this class consider more complex constraints and objectives inspired from real life applications.

On the other hand, we are interested in the multi-objective optimization approach. Such approach is so relevant, especially in vehicle routing problems with profits where the aim is to maximize the total collected profit, but also minimize the total travel costs. Anther relevant application of multi-objective optimization appears in the STOPTW, where it may be interesting to optimize both collected profit and workload balance. Another interesting approach is the use ILP solver within heuristic schemes in order to produce solution of good quality in a limited computational times. These methods are generally called math-heuristics.

Bibliographie

[IBM, 2018] (2018). IBM ILOG CPLEX optimization studio v12.6. `https://www.ibm.com/support/knowledgecenter/SSSA5P`. Accessed : 2018-09-11.

[Aho et al., 1972] Aho, A. V., Garey, M. R., and Ullman, J. D. (1972). The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2) :131–137.

[Angelelli et al., 2014] Angelelli, E., Archetti, C., and Vindigni, M. (2014). The clustered orienteering problem. *European Journal of Operational Research*, 238(2) :404–414.

[Archetti et al., 2018] Archetti, C., Carrabs, F., and Cerulli, R. (2018). The set orienteering problem. *European Journal of Operational Research*, 267(1) :264–272.

[Archetti et al., 2007] Archetti, C., Hertz, A., and Speranza, M. G. (2007). Metaheuristics for the team orienteering problem. *Journal of Heuristics*, 13(1) :49–76.

[Archetti et al., 2014] Archetti, C., Speranza, M. G., and Vigo, D. (2014). Chapter 10 : Vehicle routing problems with profits. In *Vehicle Routing : Problems, Methods, and Applications, Second Edition*, pages 273–297. SIAM.

[Bard et al., 2002] Bard, J. F., Kontoravdis, G., and Yu, G. (2002). A branch-and-cut procedure for the vehicle routing problem with time windows. *Transportation Science*, 36(2) :250–269.

[Barnhart et al., 1998] Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W., and Vance, P. H. (1998). Branch-and-price : Column generation for solving huge integer programs. *Operations research*, 46(3) :316–329.

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

[Bellman, 1954] Bellman, R. (1954). The theory of dynamic programming. Technical report, RAND Corp Santa Monica CA.

[Berger and Barkaoui, 2004] Berger, J. and Barkaoui, M. (2004). A parallel hybrid genetic algorithm for the vehicle routing problem with time windows. *Computers & operations research*, 31(12) :2037–2053.

[Bouly et al., 2010] Bouly, H., Dang, D.-C., and Moukrim, A. (2010). A memetic algorithm for the team orienteering problem. *4OR*, 8(1) :49–70.

[Boussier et al., 2007] Boussier, S., Feillet, D., and Gendreau, M. (2007). An exact algorithm for team orienteering problems. *4or*, 5(3) :211–230.

[Boykov and Kolmogorov, 2004] Boykov, Y. and Kolmogorov, V. (2004). An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE transactions on pattern analysis and machine intelligence*, 26(9) :1124–1137.

[Bullnheimer et al., 1999] Bullnheimer, B., Hartl, R. F., and Strauss, C. (1999). An improved ant system algorithm for thevehicle routing problem. *Annals of operations research*, 89 :319–328.

[Butt and Cavalier, 1994] Butt, S. E. and Cavalier, T. M. (1994). A heuristic for the multiple tour maximum collection problem. *Computers & Operations Research*, 21(1) :101–111.

[Butt and Ryan, 1999] Butt, S. E. and Ryan, D. M. (1999). An optimal solution procedure for the multiple tour maximum collection problem using column generation. *Computers & Operations Research*, 26(4) :427–441.

[Chao et al., 1996] Chao, I.-M., Golden, B. L., and Wasil, E. A. (1996). The team orienteering problem. *European journal of operational research*, 88(3) :464–474.

[Clausen, 1999] Clausen, J. (1999). Branch and bound algorithms-principles and examples. *Department of Computer Science, University of Copenhagen*, pages 1–30.

[Cordeau et al., 2001] Cordeau, J.-F., Laporte, G., and Mercier, A. (2001). A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational research society*, 52(8) :928–936.

[Czech and Czarnas, 2002] Czech, Z. J. and Czarnas, P. (2002). Parallel simulated annealing for the vehicle routing problem with time windows. In *euromicro-pdp*, page 0376. IEEE.

[Dang et al., 2013] Dang, D.-C., Guibadj, R. N., and Moukrim, A. (2013). An effective pso-inspired algorithm for the team orienteering problem. *European Journal of Operational Research*, 229(2) :332–344.

[Dang and Moukrim, 2012] Dang, D.-C. and Moukrim, A. (2012). Subgraph extraction and metaheuristics for the maximum clique problem. *Journal of Heuristics*, 18(5) :767–794.

[Dantzig and Ramser, 1959] Dantzig, G. B. and Ramser, J. H. (1959). The truck dispatching problem. *Management science*, 6(1) :80–91.

[Dantzig and Wolfe, 1960] Dantzig, G. B. and Wolfe, P. (1960). Decomposition principle for linear programs. *Operations research*, 8(1) :101–111.

[Defryn and Sörensen, 2017] Defryn, C. and Sörensen, K. (2017). A fast two-level variable neighborhood search for the clustered vehicle routing problem. *Computers & Operations Research*, 83 :78–94.

[Dorigo et al., 1996] Dorigo, M., Maniezzo, V., and Colorni, A. (1996). Ant system : optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1) :29–41.

[Eberhart and Kennedy, 1995] Eberhart, R. and Kennedy, J. (1995). A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*, pages 39–43. IEEE.

[El-Hajj et al., 2016] El-Hajj, R., Dang, D.-C., and Moukrim, A. (2016). Solving the team orienteering problem with cutting planes. *Computers & Operations Research*, 74 :21–30.

[Feillet et al., 2005] Feillet, D., Dejax, P., and Gendreau, M. (2005). Traveling salesman problems with profits. *Transportation science*, 39(2) :188–205.

[Feillet et al., 2007] Feillet, D., Gendreau, M., and Rousseau, L.-M. (2007). New refinements for the solution of vehicle routing problems with branch and price. *INFOR : Information Systems and Operational Research*, 45(4) :239–256.

[Feo and Resende, 1995] Feo, T. A. and Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2) :109–133.

[Fischetti et al., 1998] Fischetti, M., Gonzalez, J. J. S., and Toth, P. (1998). Solving the orienteering problem through branch-and-cut. *INFORMS Journal on Computing*, 10(2) :133–148.

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

[Garey and Johnson, 2002] Garey, M. R. and Johnson, D. S. (2002). *Computers and intractability*, volume 29. wh freeman New York.

[Gedik et al., 2017] Gedik, R., Kirac, E., Milburn, A. B., and Rainwater, C. (2017). A constraint programming approach for the team orienteering problem with time windows. *Computers & Industrial Engineering*, 107 :178–195.

[Gehring and Homberger, 1999] Gehring, H. and Homberger, J. (1999). A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In *Proceedings of EUROGEN99*, volume 2, pages 57–64. Citeseer.

[Gendreau et al., 1994] Gendreau, M., Hertz, A., and Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem. *Management science*, 40(10) :1276–1290.

[Glover and Laguna, 1998] Glover, F. and Laguna, M. (1998). Tabu search. In *Handbook of combinatorial optimization*, pages 2093–2229. Springer.

[Glover and Kochenberger, 2006] Glover, F. W. and Kochenberger, G. A. (2006). *Handbook of metaheuristics*, volume 57. Springer Science & Business Media.

[Gomory et al., 1958] Gomory, R. E. et al. (1958). Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical society*, 64(5) :275–278.

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

[Haddadene et al., 2016] Haddadene, S. R. A., Labadie, N., and Prodhon, C. (2016). A grasp× ils for the vehicle routing problem with time windows, synchronization and precedence constraints. *Expert Systems with Applications*, 66 :274–294.

[Hojabri et al., 2018] Hojabri, H., Gendreau, M., Potvin, J.-Y., and Rousseau, L.-M. (2018). Large neighborhood search with constraint programming for a vehicle routing problem with synchronization constraints. *Computers & Operations Research*, 92 :87–97.

[Holland, 1992] Holland, J. H. (1992). Genetic algorithms. *Scientific american*, 267(1) :66–73.

[Jepsen et al., 2008] Jepsen, M., Petersen, B., Spoorendonk, S., and Pisinger, D. (2008). Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, 56(2) :497–511.

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

[Kawamura et al., 1998] Kawamura, H., Yamamoto, M., Mitamura, T., Suzuki, K., and Ohuchi, A. (1998). Cooperative search based on pheromone communication for vehicle routing problems. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 81(6) :1089–1096.

[Ke et al., 2008] Ke, L., Archetti, C., and Feng, Z. (2008). Ants can solve the team orienteering problem. *Computers & Industrial Engineering*, 54(3) :648–665.

[Khachiyan, 1979] Khachiyan, L. G. (1979). A polynomial algorithm in linear programming. In *Doklady Academii Nauk SSSR*, volume 244, pages 1093–1096.

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, 220(4598) :671–680.

[Laporte, 1992] Laporte, G. (1992). The traveling salesman problem : An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2) :231–247.

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

[Lourenço et al., 2003] Lourenço, H. R., Martin, O. C., and Stützle, T. (2003). Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer.

[Michallet et al., 2014] Michallet, J., Prins, C., Amodeo, L., Yalaoui, F., and Vitry, G. (2014). Multi-start iterated local search for the periodic vehicle routing problem with time windows and time spread constraints on services. *Computers & operations research*, 41 :196–207.

[Mitchell, 2002] Mitchell, J. E. (2002). Branch-and-cut algorithms for combinatorial optimization problems. *Handbook of applied optimization*, pages 65–77.

[Mladenović and Hansen, 1997] Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & operations research*, 24(11) :1097–1100.

[Moscato et al., 2004] Moscato, P., Cotta, C., and Mendes, A. (2004). Memetic algorithms. In *New optimization techniques in engineering*, pages 53–85. Springer.

[Nguyen et al., 2012] Nguyen, V.-P., Prins, C., and Prodhon, C. (2012). A multi-start iterated local search with tabu list and path relinking for the two-echelon location-routing problem. *Engineering Applications of Artificial Intelligence*, 25(1) :56–71.

[Osman, 1993] Osman, I. H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of operations research*, 41(4) :421–451.

[Papadimitriou and Steiglitz, 1998] Papadimitriou, C. H. and Steiglitz, K. (1998). *Combinatorial optimization : algorithms and complexity*. Courier Corporation.

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

[Pisinger and Ropke, 2010] Pisinger, D. and Ropke, S. (2010). Large neighborhood search. In *Handbook of metaheuristics*, pages 399–419. Springer.

[Poggi et al., 2010] Poggi, M., Viana, H., and Uchoa, E. (2010). The team orienteering problem : Formulations and branch-cut and price. In *OASIcs-OpenAccess Series in Informatics*, volume 14. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[Prins, 2004] Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12) :1985–2002.

[Prins et al., 2014] Prins, C., Lacomme, P., and Prodhon, C. (2014). Order-first split-second methods for vehicle routing problems : A review. *Transportation Research Part C : Emerging Technologies*, 40 :179–200.

[Resende and Pardalos, 2008] Resende, M. G. and Pardalos, P. M. (2008). *Handbook of optimization in telecommunications*. Springer Science & Business Media.

[Roozbeh et al., 2018] Roozbeh, I., Ozlen, M., and Hearne, J. W. (2018). An adaptive large neighbourhood search for asset protection during escaped wildfires. *Computers & Operations Research*, 97 :125–134.

[Ropke and Pisinger, 2006] Ropke, S. and Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4) :455–472.

[Rossi et al., 2006] Rossi, F., Van Beek, P., and Walsh, T. (2006). *Handbook of constraint programming*. Elsevier.

[Rousseau et al., 2013] Rousseau, L.-M., Gendreau, M., and Pesant, G. (2013). The synchronized dynamic vehicle dispatching problem. *INFOR : Information Systems and Operational Research*, 51(2) :76–83.

[Souffriau et al., 2010] Souffriau, W., Vansteenwegen, P., Berghe, G. V., and Van Oudheusden, D. (2010). A path relinking approach for the team orienteering problem. *Computers & operations research*, 37(11) :1853–1859.

[Taillard, 1993] Taillard, É. (1993). Parallel iterative search methods for vehicle routing problems. *Networks*, 23(8) :661–673.

[Tarjan, 1972] Tarjan, R. (1972). Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2) :146–160.

[Toth and Vigo, 2014] Toth, P. and Vigo, D. (2014). *Vehicle routing : problems, methods, and applications*. SIAM.

[Turing, 1937] Turing, A. M. (1937). On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1) :230–265.

[van der Merwe et al., 2014] van der Merwe, M., Minas, J. P., Ozlen, M., and Hearne, J. W. (2014). A mixed integer programming approach for asset protection during escaped wildfires. *Canadian Journal of forest research*, 45(4) :444–451.

[van der Merwe et al., 2017] van der Merwe, M., Ozlen, M., Hearne, J. W., and Minas, J. P. (2017). Dynamic rerouting of vehicles during cooperative wildfire response operations. *Annals of Operations Research*, 254(1-2) :467–480.

[Vansteenwegen et al., 2011] Vansteenwegen, P., Souffriau, W., and Van Oudheusden, D. (2011). The orienteering problem : A survey. *European Journal of Operational Research*, 209(1) :1–10.

[Vargas et al., 2017] Vargas, L., Jozefowiez, N., and Ngueveu, S. U. (2017). A dynamic programming operator for tour location problems applied to the covering tour problem. *Journal of Heuristics*, 23(1) :53–80.

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

[Vidal et al., 2015] Vidal, T., Maculan, N., Ochi, L. S., and Vaz Penna, P. H. (2015). Large neighborhoods with implicit customer selection for vehicle routing problems with profits. *Transportation Science*, 50(2) :720–734.

[Yu et al., 2009] Yu, B., Yang, Z.-Z., and Yao, B. (2009). An improved ant colony optimization for vehicle routing problem. *European journal of operational research*, 196(1) :171–176.