GRASP-ILS and set cover hybrid heuristic for the synchronized team orienteering problem with time windows

Ala-Eddine Yahiaoui, Aziz Moukrim, Mehdi Serairi

GRASP-ILS and set cover hybrid heuristic for the synchronized team orienteering problem with time windows

Ala-Eddine Yahiaoui*, Aziz Moukrim, Mehdi Serairi

[a]*Sorbonne universités, Université de technologie de Compiègne*
Heudiasyc, UMR CNRS 7253, CS 60 319, 60 203 Compiègne cedex

Abstract

Wildfires are a natural phenomenon that occurs regularly in many terrestrial ecosystems. Due to global warming, the rate and the span of wildfires have remarkably increased during the last years, causing important economic losses as well as human casualties. In the last years, several initiatives have been undertaken in order to apply operations research tools to help firefighting teams schedule and optimize their protection activities when fighting wildfires.

In this context, a recent variant of the Team Orienteering Problem (TOP) called the Asset Protection Problem (APP) has been proposed in [21]. In this problem, firefighting teams should provide a protective service to a set of assets endangered by wildfires. These activities can be performed by a heterogeneous fleet of vehicles and should occur within specific time intervals estimated based on the firefronts progression. This variant of the TOP incorporates three additional constraints, namely, time windows, synchronized visits and compatibility constraints between vehicles and assets.

In this paper, we propose a hybrid approach that combines a Greedy Randomized Adaptive Search Procedure coupled with an Iterated Local Search (GRASP×ILS) and a post-optimization phase that consists of a Set Covering Problem formulation to efficiently solve the problem. Interestingly, the GRSAP×ILS incorporates an adaptive candidate-list based insertion heuristic as well as a Variable Neighborhood Descent (VND) procedure. The post optimization phase is used at the end of the resolution to improve the best solution found by the GRSAP×ILS. Detailed computational tests have been carried out to prove the effectiveness of our method. The results show that our method, called Hybrid-Heuristic, out-performs the methods of the literature, since it achieves an overall improvement of the best known solutions by 8.52% and 7.8% on medium and large-size instances respectively, while maintaining much lesser computational times.

*Corresponding author
Email address: alaeddine.yahiaoui.fr@gmail.com (Ala-Eddine Yahiaoui)

Keywords: Team orienteering problem· synchronization· greedy randomized adaptive search procedure· set covering

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. Recently, many countries have witnessed the devastating impact of wildfires on the nature and human activities: Portugal (2017), Australia (2019-2020), Sweden (2018), California (2020). 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 infrastructures. However, several challenging tasks and difficulties complicate the working environment of IMT in which they must make critical and complex decisions. Thus, the application of operations research methods, either exact or heuristic, can assist the management of wildfires in such hostile situations.

In this context, the authors in Van Der Merwe et al. [21] consider the case of an out of control wildfire 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. Moreover, IMT activities should not take action 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 cooperate together in a timely manner to carry out protection activities.

Van Der Merwe et al. [21] modeled this problem using a variant of the Team Orienteering Problem (TOP) with additional constraints, namely, time windows, synchronized visits and compatibility constraints between vehicles and assets. They consider a fleet of heterogeneous vehicles available to visit a number of strategic assets located over a geographical area. Each asset is associated with a time window within which the protection activities should start. An asset is also assigned a service time duration which represents 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 collect a given profit, the associated asset must be visited by the required vehicles in a synchronized manner, i.e. the visits performed by the vehicles should start simultaneously within the corresponding time window. Finally, the objective function of the problem 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 TOP is a class of Vehicle Routing Problems (VRP) that was proven to be NP-Hard [9]. Several variants of the TOP have been introduced in order to solve different issues related to disaster management in general. Baffo et al. [4] proposed a variant of the TOP called the Multi-Origins Capacitated Team Orienteering Problem. This variant has been used to model rescue operations during emergency situations of persons suffering problems of mobility. Balcik [5] studied a problem called the Selective Assessment Routing Problem (SARP) to address the rapid needs assessment decisions after disasters. In this variant, sites are grouped into subsets called communities, where the sites belonging to the same community share a common characteristic. A single site can have several characteristics, and hence, can be part of more than one community. The objective in this problem is to maximize the minimum coverage of each community. The coverage of a given community is computed as a ratio of the number of visited sites divided by the total number of sites of that community. A variant of SARP with uncertain travel times has been proposed in [6]. The authors proposed a resolution approach based on a robust optimization formulation with a co-axial box uncertainty set.

A major characteristic of the STOPTW is the temporal synchronization constraints. These constraints are common between several variants of VRP proposed for different real life applications. Bredström and Rönnqvist [8] studied a variant of VRP with synchronized visits in the context of home-care services. The same problem was addressed by Parragh et al. [19] and used to solve real problems related to technicians routing and scheduling problems. Crainic et al. [10] presented a VRP variant for city logistics that includes, in addition to two-echelon network architecture and multi-trip routes, an exact synchronization constraints between vehicles of first and second echelons at the level of satellites. We refer to Drexl [11] for a detailed survey on synchronization in vehicle routing problems.

Regarding the resolution approaches for the STOPTW, Van Der Merwe et al. [21] introduced a mixed integer programming model for the problem, which was demonstrated on a realistic wildfire scenario in Tasmania. The authors in [20] have proposed an adaptive large neighborhood search heuristic (ALNS) for the problem along with a new set of benchmark instances. Recently, a spatial decomposition math-heuristic (SDM) resolution approach has been proposed by Nuraiman et al. [18] for the same problem. This method succeeded to substantially improve the best solution of all medium-size and large-size instances compared with [20]. In the same vain, the authors in [22] have 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 Hybrid-Heuristic method composed of a Greedy Randomized Adaptive Search Procedure (GRASP) coupled with an Iterated Local Search (ILS) and a set covering formulation. The GRASP×ILS incorporates an adaptive candidate-list based insertion and a Variable Neighborhood

Descent (VND) search procedure used to improve the solutions produced by the ILS. The set covering problem formulation is a post-optimization phase used to extract the best solution from a pool of feasible routes previously populated by GRASP×ILS.

The remainder of this paper is organized as follows. A mathematical formulation is presented for the STOPTW in Section 2. The Hybrid-Heuristic structure is detailed in Section 3. Computational tests carried out on the methods proposed in this paper are described in Section 4. Finally, a conclusion and some perspectives are given in Section 5.

2. Problem description and mathematical formulation

The STOPTW is modeled using a directed graph $G = (V, A)$ where $V = \{0, 1, \ldots, n+1\}$ is the set of vertices and $A = \{(i, j) : i, j \in V, i \neq j\}$ is the set of arcs. $V^- = \{1, \ldots, n\}$ represents the set of assets whereas vertices 0 and $n + 1$ represent, respectively, the departure and the arrival depots. For convenience, we define two other index sets, $V^d = \{0, 1, \ldots, n\}$ and $V^a = \{1, \ldots, n, n + 1\}$. Heterogeneous fleet of vehicles is available to protect the assets. $|Q|$ types of vehicles are considered with P_q available vehicles for each type $q \in Q$. A scalar t_{ijq} is used to represent the travel time necessary for a vehicle of type $q \in Q$ to traverse arc $(i, j) \in A$. We assume that $t_{iiq} = \infty, i \in V^-, q \in Q$ and $t_{0,n+1,q} = 0, q \in Q$.

Each asset $i \in V^-$ is associated with the following data:

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

Before detailing the mathematical formulation, we first introduce the following decision variables.

- y_i: binary decision variable, it takes the value 1 if asset $i \in V^-$ is protected, 0 otherwise.
- s_i: real decision variable in $[o_i, c_i]$ associated with each asset $i \in V^-$, at which the service must start in order to protect asset i.
- z_{ijq}: binary decision variable, equal to 1 if the arc $(i, j) \in A$ 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 $(i, j) \in A$.

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

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

$$\sum_{i \in V^a} x_{0iq} = \sum_{j \in V^d} x_{j,n+1,q} = P_q \quad \forall q \in Q \tag{2}$$

$$\sum_{j \in V^d} x_{jiq} = \sum_{h \in V^a} x_{ihq} \quad \forall q \in Q, \forall i \in V^- \tag{3}$$

$$\sum_{j \in V^d} x_{jiq} = r_{iq} y_i \quad \forall q \in Q, \forall i \in V^- \tag{4}$$

$$x_{ijq} \leq P_q z_{ijq} \quad \forall q \in Q, \forall (i,j) \in A \tag{5}$$

$$z_{ijq} \leq x_{ijq} \quad \forall q \in Q, \forall (i,j) \in A \tag{6}$$

$$s_i + t_{ijq} + a_i - s_j \leq M(1 - z_{ijq}) \quad \forall q \in Q, \forall (i,j) \in A \tag{7}$$

$$o_i \leq s_i \leq c_i \quad \forall i \in V^- \tag{8}$$

$$x_{ijq} \in \{0, 1, 2, \ldots, P_q\} \quad \forall q \in Q, \forall (i,j) \in A \tag{9}$$

$$y_i \in \{0, 1\} \quad \forall i \in V^- \tag{10}$$

$$z_{ijq} \in \{0, 1\} \quad \forall q \in Q, \forall (i,j) \in A \tag{11}$$

The objective function (1) is to maximize the total collected profit. Constraints (2) ensure that all the P_q available vehicles start from the departure depot and end at the arrival depot. Constraints (3) impose that the number of incoming and outgoing vehicles is the same at each asset and for each vehicle type. Constraints (4) ensure that if a customer is served, then all of its requirements in terms of number of vehicles are met. Constraints (5) limit the capacity of each arc of type q to at most P_q vehicles. Constraints (6) are coupling constraints between z and x variables. Constraints (7) guarantee the connectivity of each tour whereas constraints (8) are the time windows constraints. Constraints (9),(10) and (11) are domain definitions.

Since the STOPTW covers the classical TOPTW as a special case, it is known to be highly complex NP-hard problem. The proposed formulation only succeeds to solve small-size instances of the problem. Using the commercial solver Cplex, only instances with at most 34 assets have been successfully solved in [20]. Due to this observation, we propose in the following the Hybrid-Heuristic to tackle medium and large-size instances.

3. Solution approach

The Greedy Randomized Adaptive Search Procedure (GRASP) is a multi-start local search metaheuristic introduced by Feo and Resende in [12]. 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 by Lourenço et al. in [16]. The basic idea behind this method is to construct in each iteration a new solution using an embedded greedy heuristic, but instead of starting each time from scratch or from a random solution, the embedded heuristic uses the solution of the previous iteration as starting solution after undergoing a perturbation phase. The series of local optima produced by this process can be seen as a single chain of solutions 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 to many vehicle routing problems such as, the Workforce Scheduling and Routing Problems [23], the Traveling Repairman Problem with Profits [3], the Periodic VRP with Time Windows [17], as well as a VRP with Synchronization and Precedence Constraints [14]. In this section, we present our GRASP×ILS global framework to solve the STOPTW. We note that in addition to the ILS, a Variable Neighborhood Descent (VND) search procedure is also embedded inside the GRASP and called every time after the ILS. The ILS uses in our case an *adaptive candidate list-based insertion* (see Section 3.2). Whereas the VND combines several local search operators in addition to the candidate list-based insertion and aims at improving both, the travel times of routes and the total collected profit.

3.1. General flow

The GRASP×ILS metaheuristic is sketched in Algorithm 1. The outer loop describes the structure of the GRASP in which $IterMax_G$ initial solutions are generated from scratch using an adaptive candidate list-based insertion. Each initial solution S is then improved using an iterative local search ILS (lines 8-21). The ILS incorporates a perturbation phase (line 12) after which a repair phase is performed using the adaptive candidate list-based insertion. S_{best} is updated every time a new best solution is found (line 15). The process is completed after $IterMax$ iterations without improvement. The solution produced by the ILS is improved by the VND and then stored in a pool of solutions $\mathcal{S}$ (lines 22-23). A post-optimization phase consists in the construction of a pool of routes $\mathcal{P}$ from $\mathcal{S}$ (line 24) and then solving a side constrained set covering problem (line 25) in order to extract the best solution. A detailed description is given in Section 3.4.

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 (comprised between 1 and d_{max}) are randomly selected and removed

from the current solution. The perturbation parameter d_{max} is initialized at 3 and incremented after each iteration without improvement (line 20). Once a new best solution is found, the perturbation parameter is reset to 3 (line 17). 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.

Algorithm 1: HYBRID-HEURISTIC ALGORITHM

```
1  S_best ← ∅
2  for (i = 1; i < IterMax_G; i++) do
3      (α, β, γ) ← (0.5, 0.5, 0.5)
4      S ← ∅
5      (S, α, β, γ) ← AdaptiveInsertion(S, α, β, γ) (see Section 3.2)
6      S_ILS ← S
7      S ← S ∪ {S}
8      Iter ← 0
9      d_max ← 3
10     while Iter < IterMax do
11         d ← U(1, d_max)
12         Remove d assets from S
13         Update S
14         (S, α, β, γ) ← AdaptiveInsertion(S, α, β, γ) (see Section 3.2)
15         if (Profit(S) > Profit(S_ILS)) then
16             S_ILS ← S
17             d_max ← 3
18             Iter ← 0
19         else
20             d_max ← d_max + 1
21             Iter ← Iter + 1
22     S_VND ← VariableNeighborhoodDescent(S_ILS) (see Section 3.3)
23     S ← S ∪ {S_VND}
24     if (Profit(S_VND) > Profit(S_best)) then
25         S_best ← S_VND
26 P ← InitPool(S)
27 S_c ← setCover(P) (see Section 3.4)
28 if (Profit(S_c) > Profit(S_best)) then
29     S_best ← S_c
30 return S_best
```

3.2. Adaptive candidate list-based insertion

The main component of ILS is the insertion algorithm. This algorithm starts from an initial solution, which can be empty, and add unrouted assets one by one. The insertion process stops when all the assets are inserted or no more insertions are possible. Before starting the insertion process, the unrouted assets are first sorted according to non-decreasing values of what we call the insertion criterion. We consider in this criterion the following factors:

- The profit. Since the objective function is to maximize the collected profit, this criterion favors the assets with higher values of profit to be inserted.
- The width of 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.
- 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 normalize each factor by its maximum possible value so that all the factors in the criterion have their values within the interval $[0, 1]$. To that end, let t_{max} be the width of the largest time window, p_{max} be the highest profit among all the assets and r_{max} be the maximum number of vehicles required by the assets.

Hence, the insertion criterion is calculated for each asset as follows:

$$Cr_i(\alpha, \beta, \gamma) = \frac{(\frac{c_i - o_i}{t_{max}})^{\beta}}{(\frac{p_i}{p_{max}})^{\alpha}(\frac{|r_i|}{r_{max}})^{\gamma}} \tag{12}$$

As shown in equation (12), 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 [7]. 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 determine 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 steps of 0.1. This results in four different combinations of (β,γ). In the fifth function f_5, β and γ are randomly generated using a uniform distribution in the interval $[0, 1[$, while α is set to 1. Parameter α is always set to 1 in order to favors the insertion of assets with higher profits, since the aim of solving the STOPTW is to maximize the total collected profit. However, we consider a last function where all the parameters are randomly generated within

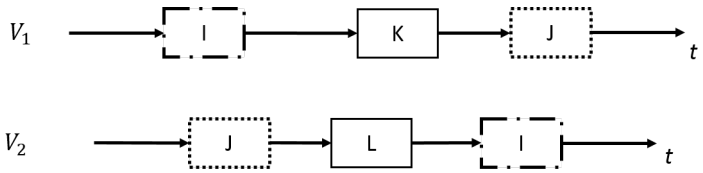

Figure 1: Cross-synchronization situation

$[0, 1[$. At the end of each iteration of the ILS, the combination that led to the solution with the best collected profit is used as a basis for the next iteration.

In order to accelerate the insertion process and avoid non-feasible moves, we present the following two mechanisms to deal with issues related to synchronization constraints and time windows. Before proceeding further, let us first define a feasible solution S as a set of routes $R = \{R_1, R_2, ..., R_{|R|}\}$ performed by distinct vehicles. Each route $R_k \in R$ is composed of an ordered list of visits that starts from the departure depot and ends at the arrival depot.

We denote by $R_k(p)$ the p^{th} asset visited in route R_k. We also denote by $R^i \subseteq R$ the set of routes that visit the asset $i \in V^-$.

3.2.1. Preventing cross-sychronization

A key characteristic in the STOPTW is that the routes are interdependent, and hence, any update needs to be propagated through all the routes of the solution after each insertion or partial destruction. However, the propagation may loop infinitely if cross synchronization is not prohibited. Without loss of generality, given two arbitrary assets i and j that require two vehicles with exactly the same types, a situation of cross synchronization happens when a first vehicle visits asset j before i, while a second vehicle visits asset i before j. An example of cross-synchronization situation is illustrated by figure 3.2.1. To avoid such situations, we proceed in a similar way as Afifi et al. [1].

Let us consider an auxiliary graph that we call the precedence graph $H = (V, X)$ associated with a feasible solution S, where V has the same set of nodes as G and X is the arc set. X is first initialized with a set of precedence relations based on the arcs that relates between each successive visits in S, i.e. if asset j is visited immediately after asset i in a given route $R_k \in R$, a precedence relation (an arc) from asset i to asset j is added to X. Then, the precedence relations are propagated beyond immediate successors using the transitive closure [2]. A detailed description is given in algorithms 2 and 3. The algorithm construct the set of reachable assets of each asset asset by launching a Depth First Search (DFS) from each node $i \in G$. The array *marked* is used to mark visited nodes in order to avoid revisiting them during the current DFS.

The construction of the precedence graph from scratch yields to $O(n(n + E))$ time complexity, where DFS algorithms ($\mathcal{O}(n + E)$ time complexity) are launched from each node. E being the total number of arcs at the end of the graph construction, with worst case value equal to n^2. The sets $\Gamma^+(i)$ defines

Algorithm 2: Construction of precedence graph

Input: *Adjacency Lists* Γ^+

for $i = 0$ *to* n **do**
  $InitArray(marked, n, 0)$
  $DFS(\Gamma^+, i, visited, i)$

Algorithm 3: Depth Frist Search (DFS)

Input: *Adjacency Lists* Γ^+, Root node i, *Array* marked, Current node j

$marked[j] = 1$
foreach $elet \in \Gamma^+(j)$ **do**
  if $i \neq elet$ and $marked[elet] = 0$ **then**
    $\Gamma^+(i) \leftarrow \Gamma^+(i) \cup \{elet\}$
    $DFS(\Gamma^+, i, visited, elet)$

for each vertex $i \in H$, the set of all of its successors in H. Once the precedence graph H is completed along with the adjacency lists Γ^+, an adjacency matrix Ψ is constructed in $\mathcal{O}(n^2)$ to allow an access to precedence relations in a constant time. A precedence relation from asset i to asset j is simply represented in Ψ by setting $\Psi[i][j] = 1$.

The cross-synchronization feasibility check is described as follows. The following proposition holds.

Proposition 3.1. *The insertion of a visit of a given asset x between two visits of assets i and j in an arbitrary route $R_k \in R$ does not create a situation of cross-synchronization if and only if: $i \notin \Gamma^+(x)$ and $x \notin \Gamma^+(j)$*

Proof. $\Rightarrow$) Assume that $i \in \Gamma^+(x)$ or $x \in \Gamma^+(j)$. If x is inserted after a visit of i and before a visit of asset j, we have then $x \in \Gamma^+(i)$ and $j \in \Gamma^+(x)$. Hence, a situation of cross-synchronization happens between either i and x or between j and x.
$\Leftarrow$) Assume that there is a cross-synchronization after the insertion of a visit of asset x between visits of asset i and j. On the other hand, after the insertion of x, we have $x \in \Gamma^+(i)$ and $j \in \Gamma^+(x)$. However, we have already assumed that $i \notin \Gamma^+(x)$ and $x \notin \Gamma^+(j)$. □

This test is performed before the insertion of visits of assets that require multiple vehicles. Using the adjacency matrix Ψ, this feasibility check is performed in a constant time ($\mathcal{O}(1)$).

The construction of the precedence graph H and adjacency matrix Ψ from scratch is performed after each perturbation phase (partial destruction). However, only an update of Ψ is needed after the insertion of a new visit. The update is performed as follows. Let us consider the insertion of a visit of asset i

in a given route after a visit of asset j and before a visit of asset k. The update of Ψ should respect the following rules:

- $\Psi[j][i] \leftarrow 1$ and $\Psi[i][k] \leftarrow 1$
- $\Psi[i][i\prime] = 1 \Rightarrow \Psi[j][i\prime] \leftarrow 1$
- $\Psi[j\prime][j] = 1$ and $\Psi[i][i\prime] = 1 \Rightarrow \Psi[j\prime][i\prime] \leftarrow 1$
- $\Psi[k][k\prime] = 1 \Rightarrow \Psi[i][k\prime] \leftarrow 1$
- $\Psi[i\prime][i] = 1$ and $\Psi[k][k\prime] = 1 \Rightarrow \Psi[i\prime][k\prime] \leftarrow 1$

As a result, the update of the adjacency matrix Ψ after each insertion is performed in $\mathcal{O}(n^2)$ space and time complexity.

3.2.2. Time-window feasibility check

Time windows feasibility check is a frequent operation performed several times before each insertion. An efficient way to verify the feasibility in a constant time is provided in Afifi et al. [1]. The authors proposed to keep some useful information in data structures and update it after each insertion or partial destruction. An assumption is made that no cross-synchronization situations exist in the current solution S.

Before going further, here some useful notation:

- Str_i is the service starting time of $i \in V$. Str_0 is set to 0.
- $Maxshift_i$ is the maximal delay allowed for the service of $i \in V$. $Maxshift_{n+1}$ is set to $+\infty$.
- $Maxshift_i^k$ is the maximal delay allowed for the service of i in route $R_k \in R$ and regardless the synchronization with the other visits of i in other routes. Note that $Maxshift_{n+1}^k$ is set to $+\infty$, $k : R_k \in R$.
- Arr_i^k is the arrival time at asset i in route $R_k \in R$.
- $Wait_i^k$ is the waiting time at asset i in route $R_k \in R$. $Wait_{n+1}^k$ is set to 0, $k : R_k \in R$.

Due to synchronization constraints, the starting time service should be delayed so that all the assigned vehicles are present at the asset location. It holds that:

$$Str_i = max\{o_i, \max_{k:R_k \in R^i}\{Arr_i^k\}\} \quad i \in V^- \tag{13}$$

Following this, the waiting time at asset $i \in V^-$ in route $R_k \in R^i$ is:

$$Wait_i^k = Str_i - Arr_i^k, \quad k : R_k \in R^i \tag{14}$$

In order to calculate the $Maxshift_i$ at asset $i \in V^-$, we first compute different $Maxshift_i^k$ for each route $R_k \in R^i$:

$$Maxshift^k_{i=R_k(p)} = min\{c_i - Str_i, Wait^k_{R_k(p+1)} + Maxshift_{R_k(p+1)}\} \quad k : R_k \in R^i \tag{15}$$

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

$$Maxshift_i = \min_{k:R_k \in R^i} \{Maxshift^k_i\} \tag{16}$$

On the other hand, if a visit of an asset x gets to be inserted in route $R_k \in R$ between visits at positions p and $p+1$, the generated shift ($Shift^{k,p}_x$) is calculated as:

$$Shift^{k,p}_x = t_{R_k(p)x} + Wait^k_x + a_x + t_{x,R_k(p+1)} - t_{R_k(p),R_k(p+1)} \tag{17}$$

Hence, the following proposition holds.

Proposition 3.2. *The insertion of a visit of an asset x between positions p and $p+1$ is feasible in terms of time windows if:*
$Str_{R_k(p)} + a_{R_k(p)} + t_{R_k(p)x} \leq c_x$ ***and*** $Shift^{k,p+1}_x \leq Wait^k_{k(p+1)} + Maxshift_{k(p+1)}$.

We note that computing service starting times and maximum delays of a solution S is equivalent to computing the longest path in an acyclic graph. Basically, this procedure consists of a two phases : a forward and backward passes. The forward pass is performed to compute the earliest starting times Str_i at each node i, whereas the backward pass is used to calculate the maximum delays $MaxShift_i$. Hence, the complexity of updating this information is $\mathcal{O}(n+E)$ where n is the number of assets and E is the number of arcs. Whereas, the time-window feasibility test is in $\mathcal{O}(1)$.

3.2.3. Candidate list-based insertion algorithm

The candidate list-based insertion is described in Algorithm 4. It starts by sorting the unrouted assets in non-decreasing order of the criterion presented by the formula 12 (line 1). In each iteration of the insertion process (line 3), the first available asset in the candidate list is selected, and for each required type of resources/vehicles, a list of potential insertion positions are computed and stored in Δ (line 9). Then, the algorithm randomly selects one position and checks its feasibility. First, it checks for the *cross-synchronization* feasibility (line 15). We note that this test is not performed when trying to insert the first visit of an asset in the current solution, since the problem of cross synchronization only occurs with the presence of two or more visits. The second test the *time-windows* feasibility check (line 16). If the position does not satisfy one of the feasibility tests, it is then removed from Δ (line 22). This process is repeated until a feasible position is found, or the set of potential positions Δ is empty. If a feasible position is found, the insertion of the visit is performed (line 17) and the solution undergoes an update (line 18) in order to add the new precedence relations in the precedence graph as well as computation of new service start times and maximum delays associated with each asset in the solution. This process is reiterated as many times as the number of required visits.

If at the end the requirements of a given asset are not fully satisfied, all the visits of that asset are completely removed from the solution (lines 27-29) and the solution S undergoes an update in order to remove the irrelevant precedence relations and also to recompute service start times and maximum delays. The whole process of insertion of assets is reiterated until all the candidates in σ are tested.

3.3. Variable neighborhood decent

The *candidate-list based insertion* operator and its adaptive criterion feature allow our method to cover a large part of the search space. However, these components discard a key performance measure which is the travel time incurred by insertions. The reason why the travel times is not incorporated in the insertion criterion is that it will substantially increase computational complexity during the construction of the candidate-list, since the insertion cost of all unrouted assets for all positions should be calculated ($\mathcal{O}(n^2)$). Whereas, the profit, the width of time windows and the resource requirements, are all known beforehand. The Variable Neighborhood Decent (VND) described hereafter is precisely proposed in order to cope with this drawback.

Several local search operators were implemented and incorporated in the VND search procedure. Two sets of neighborhoods are considered. The first set includes operators that aim at reducing the travel times, which are:

1. *2-opt** — interchanges the tails between two routes
2. *relocate* — Relocate one visit to another position in another route

If the two first operators succeed to reduce the travel times of the solution, this may create room to insert profitable assets in the solution, and hence improving the total profit. The second set operators are :

1. *replacement* — exchanges routed assets with unrouted assets, only if it improves the total profit
2. *candidate list-based insertion* — described earlier

Seeking for efficiency, the previously described operators (*relocate* and *replacement*) are implemented with a reduced-size neighborhoods. Due to the synchronization constraints, checking the feasibility of some moves in the original version of these operators requires removing then inserting subsets of customers. This yields to $\mathcal{O}(n^2)$ time complexity in order to check the feasibility of each move. In our case, we only consider the moves for which the feasibility can be checked at a constant time $\mathcal{O}(1)$. Regarding *Replacement* operator, the reduced neighborhood is described as follows. Let us consider the replacement of asset u by an unrouted asset v. Such move is considered only if $p_v > p_u$ and $r_{uq} \leq r_{vq} \quad \forall q \in Q$. For the *Relocate operator*, we only consider assets with one required visit and a relocation to a different route than the current one. It is also noteworthy to mention that for the neighborhoods *2-opt** and *candidate list-based insertion*, feasibility check is already performed in $\mathcal{O}(1)$ for all the moves.

Algorithm 4: LIST-BASED INSERTION

```
Input: Solution S, parameters (α,β,γ)
σ ← Sort unrouted assets to non-decreasing values of Cr_i(α,β,γ) (See Section 3.2)
insert ← True
while insert and σ ≠ ∅ do
    insert ← False
    for i = 1 to |σ| do
        k_max ← |r_σi|
        k ← 0
        for q = 1 to |Q| do
            Δ ← list of all positions in routes of type q ∈ Q of S
            cpt ← 1
            while cpt ≤ r_σiq do
                foundPos ← False
                while foundPos = False and Δ ≠ ∅ do
                    Select a random position (q,r,p) ∈ Δ  // r : a route of S of type q ∈ Q, p : position in r
                    if k = 0 or (q,r,p) is cross-synchro-feasible for σ_i then
                        if (q,r,p) is time-window-feasible for σ_i then
                            Insert σ_i in position (q,r,p)
                            Update S
                            foundPos ← True
                            cpt ← cpt + 1
                            k ← k + 1
                    Δ ← Δ\{(q,r,p)}
                if foundPos = False then
                    break                // no feasible position in Δ
            if cpt ≤ r_σiq then
                break     // requirements of type q are not fully satisfied
        if (k < k_max) then
            Remove all visits of σ_i from S
            Update S
        else insert ← True
return S
```

Algorithm 5 describes the VND in details. The set $N^l = \{N_1^l, N_2^l\}$ respectively contains *replacement* and *relocate* operators, whereas the set $N^p = \{N_1^p, N_2^p\}$ respectively contains *2-opt** and *candidate list-based insertion* operators. The

first level of VND (lines 3-10) tries to improve the travel times, then the second level (lines 11-19) focuses on improving the profit. Neighborhoods are applied sequentially one by one in each level. Each level of VND is executed iteratively until no neighborhood succeeds to improve the solution. In addition, if the second level neighborhoods succeed to improve the profit, the whole VND starts a new iteration, otherwise, the algorithm is terminated.

Algorithm 5: VARIABLE NEIGHBORHOOD DECENT

```
   Input: Solution S
 1 impr ← True
 2 k^1_max, k^2_max ← |N^l|, |N^p|
 3 while impr = True do
 4     imprTravelTime ← True
 5     while imprTravelTime = True do
 6         imprTravelTime ← False
 7         for k^1 = 1 to k^1_max do
 8             S' ← N^l_{k^1}(S)
 9             if TravelTime(S') < TravelTime(S) then
10                 S ← S'
11                 imprTravelTime ← True
12     imprProfit ← True
13     while imprProfit = True do
14         imprProfit ← False
15         for k^2 = 1 to k^2_max do
16             S' ← N^p_{k^2}(S)
17             if Profit(S') > Profit(S) then
18                 S ← S'
19                 imprProfit ← True
20                 impr ← True
21 return S
```

3.4. The set covering problem

As described earlier, only local optima found throughout the search process are temporarily saved, and then rejected as soon as a new best solution is found. Those rejected solutions, despite sub-optimal, may incorporate some good individual routes. Moreover, individual routes may happen to be promising if combined with routes from other solutions and can yield new improving solutions. To that end, we propose a route recombination procedure as a post-optimization phase in order to get the best of possible solutions.

During the search process, solutions produced by the GRASP×ILS combined with the VND are stored in a set $\mathcal{S}$. At the end, single routes are extracted

from $\mathcal{S}$ and saved in a pool $\{\mathcal{T}_q|q \in Q\}$, where $\mathcal{T}_q = \{T_{q1}, T_{q2}, \ldots, T_{q|\mathcal{T}_q|}\}$. Route recombination phase consists in solving a modified set covering problem (SCP) over $\mathcal{T}_q|q \in Q$ in order to extract a combination of routes that defines the best possible solution. In the following, we propose a mixed integer programming (MIP) formulation to solve the SCP. Let first introduce some necessary notation and define the decision variables. Besides variables y and s introduced in Section 2, we use binary decision variable $\theta_k^q,\ 0 < q \leq |Q|,\ 0 < k \leq |\mathcal{T}_q|$ to indicate whether route T_{qk} is selected or not in the solution found by the solver.

We denote by singleton $\gamma_{qk}^+(i)$ the asset visited after $i \in V^-$ in route T_{qk}.

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

$$A_q = (a_{ik}^q)\ with\ a_{ik}^q = \begin{cases} 1 & if\ asset\ i \in T_{qk} \\ 0 & otherwise \end{cases}$$

The mathematical formulation, $[SCP1]$, is as follows:

$$[SCP1] \quad max \sum_{i \in V^-} p_i y_i \tag{18}$$

$$\sum_{k:T_{qk} \in \mathcal{T}_q} a_{ik}^q \theta_k^q \geq r_{iq} y_i \quad \forall q \in Q, \forall i \in V^- \tag{19}$$

$$\sum_{k:T_{qk} \in \mathcal{T}_q} \theta_k^q \leq P_q \quad \forall q \in Q \tag{20}$$

$$s_i + t_{ijq} + a_i - s_j \leq M(1 - \theta_k^q) \quad \forall q \in Q, \forall k : T_{qk} \in \mathcal{T}_q, \forall i \in T_{qk}, j \in \gamma_{qk}^+(i) \tag{21}$$

$$t_{0iq} - s_i \leq M(1 - \theta_k^q) \quad \forall q \in Q, \forall k : T_{qk} \in \mathcal{T}_q, i \in \gamma_{qk}^+(0) \tag{22}$$

$$o_i \leq s_i \leq c_i \quad \forall i \in V^- \tag{23}$$

$$y_i \in \{0, 1\} \quad \forall i \in V^- \tag{24}$$

$$\theta_k^q \in \{0, 1\} \quad \forall q \in Q, \forall\ k,\ 0 < k < |\mathcal{T}_q| \tag{25}$$

$[SCP1]$ aims at maximizing the total collected profit (18) subject to the set of constraints (19-23). Constraints (19) ensure the satisfaction of resource requirements for assets, whereas (20) impose an upper limit on the number of vehicles. It is noteworthy to mention that in this formulation, due to the inequality used in constraint (19), it is possible to visit assets more than the required number of visits. Such choice is motivated by the fact that an equality would be too restrictive and prevent $[SCP1]$ from finding good solutions.

Time constraints are initially verified by all the routes present in the pool. However, the combination of different routes can cause a violation of time constraints, such as time windows, travel times and synchronization. To avoid such issues, we add constraints (21) and (22) in which only one service starting time decision variable s_i is used per asset i. In this way, we impose that a given asset i must be visited in the selected routes at the same time. Moreover, a minimum travel time is imposed between each two consecutive visits of in selected routes, even if the their assets are not selected in the solution found by $[SCP1]$. Constraints (21) and (22) are systematically deactivated if the route is not selected. Finally, constraints (24-25) are domain definition.

An important drawback of this formulation is that if a given route $T_{qk} \in \mathcal{T}_q$ is selected in the solution found by $[SCP1]$, all the visits of its routes will be then considered. That is to say, travel time constraints between successive visits should be respected, although the visited assets are not selected in the final solution. This has been said, a cleaning phase is carried out after the resolution in order to remove the visits of non selected assets as well as unnecessary and irrelevant visits from the selected routes.

To deal with this drawback posed by $[SCP1]$, we propose in the following another formulation $[SCP2]$ that allows to skip the visits of non selected assets. Let us first denote by $\Gamma^+_{qk}(i)$ the set of assets visited in route T_{qk} after visiting $i \in V^d$. The new formulation is as follows.

$$[SCP2] \qquad 18, 19, 20, 23, 24, 25$$

$$s_i + t_{ijq} + a_i - s_j \leq M(3 - y_i - y_j - \theta^q_k) \quad \forall q \in Q, \forall k : T_{qk} \in \mathcal{T}_q, \forall i \in V^-, j \in \Gamma^+_{qk}(i) \quad (26)$$

$$t_{0iq} - s_i \leq M(2 - y_i - \theta^q_k) \quad \forall q \in Q, \forall k : T_{qk} \in \mathcal{T}_q, \forall i \in \Gamma^+_{qk}(0) \quad (27)$$

According to $[SCP2]$, a minimum travel time is imposed between each two consecutive visits of selected assets in the selected routes. Constraints (26) and (27) are systematically deactivated if at least one of the assets i or j or the route T_{qk} are not selected in the solution found by $[SCP2]$. Still, the solution found by $[SCP2]$ need a cleaning phase in order to get rid of unnecessary visits in case where a given asset is visited more than required.

The pool size is a critical performance parameter. Adding all the feasible routes to the pool may incur a long computational times. We impose in our case an upper limit on the size of the pool, which is a parameter of the algorithm to be tuned. Furthermore, in order to diversify the pool and avoid duplicated routes, we use a hash-based function. Two routes are considered as duplicates if they are from two different solutions and have the same sequence of visits. It is noteworthy to mention that it is possible that a single solution may incorporate routes with exactly the same sequence of visits. This happens when the visited assets all have similar requirements in terms of type and number vehicles. In

this case, these routes are all inserted into the pool. In this way, we ensure that the $[SCP]$ do not miss the best solution obtained no far by the GRASP×ILS. We propose also to accelerate our method by solving the LP relaxation of the MIP. If the relaxation is no better than the best solution already found, the resolution of the resolution of MIP is skipped and computational efforts are saved. This situation is frequent since the size of the pool is limited and the quality of the best solution is tightly dependent on the quality of the routes, which are exclusively generated by the GRASP×ILS and the VNS.

4. Computational tests

We investigate in this section the performance of Hybrid-Heuristic method. It was implemented using C++ and STD library, whereas the set covering formulation was solved using the IBM ILOG suite (Cplex 12.6 solver) through Concert Technology. The experimental tests were conducted on a Linux server running Centos 5.4 equipped with an Intel Xeon E5420 with 2.66 GHz and 128 GB RAM.

4.1. Benchmark instances

Benchmark instances used in [20] to evaluate STOPTW methods were generated based on 60 problem instances proposed initially for the VRPTW in [13]. These instances are divided into three classes according to the distribution of the assets over an area of 140×140. The assets in the instances are located randomly (R), in clusters (C) or combining both (RC). Each class pattern is divided into 2 sub-classes (R100-R200, C100-C200 and RC100-RC200). Finally, ten instances are derived from each combination of location and horizon patterns. Time windows of the original instances were modified in order to simulate a propagation of fire fronts across the area, whereas the requirements in terms of vehicles were randomly generated and added as a vector of three components, that is, each component corresponds to the number of vehicles of a given type (three types of vehicles). Instances in the benchmark are all composed of 200 assets in addition to the depot, and each instance was used to derive a second instance by truncating the first 100 vertices. When solving these instances, two different sets of vehicles are used for each size, namely, $SET1 = (6, 5, 4)$ and $SET2 = (7, 6, 5)$ for 100-node instances, whereas for 200-node instances, the sets are $SET1 = (9, 8, 7)$ and $SET2 = (12, 11, 10)$. As a result, the benchmark is composed of 240 instances, with 120 instances for each size (100 and 200).

4.2. Parameters setting

A key feature of our proposed method is the small number of parameters that need to be tuned. Moreover, we adopt a general approach that consists in choosing method parameters based on the parameters of the problem. In order to fine tune different parameters, we used an automatic algorithm configuration package called *iRace* software by Lopez-Ibanez et al. [15]. We selected an arbitrary set of 24 benchmark instances and gave *iRace* a tunning budget of 2000

experiments. Four parameters are considered for tuning: the number of iterations of the GRASP $IterMax_G$, the number of iterations without improvement of the ILS $IterMax$, the CPU time allocated for the set covering formulation, and finally the size of the pool $\mathcal{S}$.

Starting with the ILS, the maximum number of iterations is already fixed to $n + \lambda\overline{q}$, where $\overline{q}$ is the average number of vehicles per type calculated as $\overline{q} = \frac{\sum_{q \in Q} P_q}{|Q|}$, and λ is a weight parameter determining the influence of $\overline{q}$ on the value of $IterMax$. Regarding the GRASP, which is a set of independent iterations of the ILS, the stopping criterion is set to the average number of vehicles per type $IterMax_G = \overline{q}/\mu$, where μ is determined after tuning. Regarding the set covering problem formulation, two parameters have been considered: the size of the pool ($\mathcal{S}$) and the time limit (SCP_{time}). The time limit SCP_{time} is equal to $n\zeta$, where ζ is a constant set to be tuned. With the help of *irace* package, we suggest the parameter settings depicted in Table 1.

Table 1: Parameter setting for the Hybrid-Heuristic

Parameter	λ	μ	$\lvert S\rvert$	ζ
Value	0.25	0.5	200	0.035

4.3. Sensitivity analysis

We present in this section the sensitivity analysis of the different components proposed in this paper. The aim is to investigate the relevance of incorporating the VND and the set covering formulation within the GRASP×ILS. Different configurations have been considered and run a single time on all the benchmark instances. To clearly highlight the outcome of each component in this paper, we have used the same starting seed per instance for all the configurations. The results of each configuration are aggregated and organized by classes (C, R and RC) of the benchmark instances, where $\sum Obj$ and $\sum T(s)$, denote respectively, the sum of the total collected profit and the sum of computational times.

4.3.1. Comparison between the set covering formulations

Table 2 shows a comparison between the set covering formulations presented in section 3.4. For this purpose, three configurations are considered: GRASP×ILS, GRASP×ILS + SCP1 and GRASP×ILS + SCP2. Table 2 clearly demonstrates the performance achieved by the second SCP formulation, whether in terms of objective value or computational times. Regarding the objective value, SCP2 out-performs SCP1, this is justified by the fact that SCP1 consider systematically all the visits in every selected route of the optimal solution, which reduces substantially the space of solution, and hence, potentially missing solutions with better objective value. Interestingly, SCP2 also achieved better computational times that SCP1, with a total time of 8983.55 seconds against 11655.82 seconds.

Table 2: Comparison between the two set covering formulations

Class	GRASP×ILS+SCP1		GRASP×ILS+SCP2	
	$\sum Obj$	$\sum T(s)$	$\sum Obj$	$\sum T(s)$
C	153291	3345.51	153310	2679.18
R	159261	4013.73	159301	3121.47
RC	161714	4296.58	161739	3182.9
$\sum$	474266	11655.82	474350	8983.55

4.3.2. Analysis of components

As a result of the experimentation made in the previous section, the second formulation [$SCP2$] has been chosen to be part of the Hybrid-Heuristic. We investigate in the following the impact of different configurations of our scheme. According to Table 3, four configurations are considered: GRASP×ILS, GRASP×ILS + SCP2, GRASP×ILS + VND and finally, GRASP×ILS + VND + SCP2. The first configuration is used as a reference for the other ones. The results clearly show the contribution of the VNS and the post-optimization phase, denote by SCP2 in the Table 3. Starting with the post-optimization phase in the second column, which succeeds to improve the aggregated profit by roughly 1089 units compared to GRASP×ILS, equivalent to an improvement of 0.23%. This improvement has come at the expense of a substantial increase in computational times, reaching $8983.55s$ against $2374.96s$ for GRASP×ILS. Interestingly, we notice in column 3 that the VNS achieves a substantial improvement of the results with a gap of 2.06% compared to the GRASP×ILS, along with reasonable increase in computational times, about $4500s$, roughly half of the computational times of GRASP×ILS + SCP2. We note that the improvement made by the SCP2 is less substantial than that of the VNS. The reason behind is that the SCP2 includes time related side constraints which destroys the original set covering structure. As a result, MIP solvers struggles to solve the problem even on a small number of routes. Finally, the combination of the three components, depicted in the fourth column, achieves the best results in terms of objective values, reaching an improvement of 2.12% compared to GRASP×ILS. Regarding the computational times, the three-component configuration is better than the GRASP×ILS+SCP2, this is mainly due to the fact that the substantial improvement yielded by the VNS allows less room for the post-optimization phase to improve the objective value, which often obtains an LP relaxation equal to the best solution already found by GRASP×ILS+VNS.

4.4. Performance comparison

In this section, we conduct experimental tests to evaluate the performance of the proposed method. We compare the Hybrid-Heuristic against two methods

Table 3: Sensitivity analysis of the three components

Class	GRASP×ILS		GRASP×ILS+SCP2		GRASP×ILS+VNS		GRASP×ILS+VNS+SCP2	
	$\sum Obj$	$\sum T(s)$	$\sum Obj$	$\sum T(s)$	$\sum Obj$	$\sum T(s)$	$\sum Obj$	$\sum T(s)$
C	152914	790.72	153310	2679.18	155462	1289.73	155622	2566.01
R	158959	769.33	159301	3121.47	162383	1544.4	162460	2816.63
RC	161388	814.91	161739	3182.9	165171	1673.53	165210	2266.34
SUM	473261	2374.96	474350	8983.55	483016	4507.66	483292	7648.98

from the literature, namely, an ALNS method proposed in Roozbeh et al. [20] and a mathheuristic approach called SDM, proposed in Nuraiman [18]. To evaluate our method, we proceed in a similar way as in [20], that is, we run our algorithm ten times on each instance and we record the average computational times, the best objective value as well as the average objective value of the ten runs. As it is the same protocol, the comparison between these two methods is straightforward. On the other hand, the SDM approach as described in [18] is a deterministic method, and hence, it is executed only once for each instance. Therefore, in order to guarantee a fair comparison between the hybrid heuristic and the SDM approach, we compare the objective value obtained by the first execution of our method against the objective value of the SDM approach.

Tables 4-7 show a comparison between the three methods. The results are organized by sub-classes present the benchmark instances. In Tables 4-7, we denote the results of the first execution of our method by Hyb-Heur$_1$, whereas the results of the ten runs are denoted by Hyb-Heur$_{10}$. We provide for different methods the following performance measurements:

- $P(\%)$: the percentage of profit related to the protected assets achieved by SDM and Hyb-Heur$_1$ after one execution.
- $P_{best}(\%)$: the best percentage of profit related to the protected assets achieved by ALNS or Hyb-Heur$_{10}$ after ten runs.
- $T(s)$: computational times achieved by by SDM and Hyb-Heur$_1$ after one execution.
- $\overline{T}(s)$: average computational times achieved by a ALNS or Hyb-Heur$_{10}$ after ten runs.
- $Dev(\%)$: the deviation of the average results from the best results achieved by a ALNS or Hyb-Heur$_{10}$ after ten runs. It is calculated as follows :

$$Dev = \frac{P_{best} - \overline{P}}{P_{best}} \tag{28}$$

where $\overline{P}$ is the average percentage of profit related to the protected assets achieved by a ALNS or Hyb-Heur$_{10}$ after ten runs.

- *Gap*(%) : the last two columns of each table depict the improvement gap achieved by the Hybrid-Heuristic compared to the other two approaches. The formula used to calculated the gap between the Hybrid-Heuristic and ALNS is :

$$GAP = \frac{P_{best}^{Hyb-Heur_{10}} - P_{best}^{ALNS}}{P_{best}^{ALNS}}. \tag{29}$$

whereas the gap between the Hybrid-Heuristic and SDM is :

$$GAP = \frac{P^{Hyb-Heur_1} - P^{SDM}}{P^{SDM}}. \tag{30}$$

A positive value of the GAP means that Hybrid-Heuristic outperforms the other method.

Table 4: Comparison on 100-node instances - SET 1 with $(6, 5, 4)$ vehicles

Class	ALNS			SDM		Hyb-Heur$_{10}$			Hyb-heur$_1$		GAP (%)	
	$\overline{T}(s)$	$P_{best}(\%)$	$Dev(\%)$	$T(s)$	$P(\%)$	$\overline{T}(s)$	$P_{best}(\%)$	$Dev(\%)$	$T(s)$	$P(\%)$	ALNS	SDM
$C100$	138.47	61.84	2.77	95.45	65.09	9.21	70.20	1.02	8.42	69.19	13.52	6.29
$C200$	133.48	60.72	2.84	44.48	61.17	7.55	67.38	1.56	7.01	66.55	10.96	8.79
$R100$	134.47	62.50	2.14	44.30	63.68	7.74	70.49	1.47	7.50	69.86	12.79	9.69
$R200$	135.75	65.30	2.61	16.35	65.88	8.31	73.29	1.34	8.15	72.27	12.24	9.69
$RC100$	144.20	68.59	2.72	63.51	69.57	9.36	77.68	1.49	9.65	76.29	13.26	9.66
$RC200$	142.97	69.13	2.91	79.26	70.88	8.83	78.02	1.35	9.29	76.81	12.87	8.37
$Mean$	138.22	64.68	2.66	57.22	66.05	8.50	72.84	1.37	8.34	71.83	12.60	8.75

Tables 4 and 5 report the results of 100-node instances. These results show a clear dominance of the Hybrid-Heuristic over the other methods in terms of both, objective values and computational times. For instance, Hybrid-Heuristic achieves an improvement gap of 12.6% compared to ALNS for $(6, 5, 4)$ vehicles and a gap of 13.68% for $(7, 8, 9)$ vehicles. Compared to SDM approach, Hybrid-Heuristic achieves an improvement gap of 8.75% and 8.29% for the same sets of instances. Moreover, the Hybrid Heuristic demonstrates better robustness than the ALNS in terms of solution quality when executed several times, with a deviation from the best solution no more than 1.37% for $SET1$ and 1.3% for $SET2$ against 2.66% and 2.39% for ALNS. The Hybrid-Heuristic also achieves best computational times, since it divides by a factor of more than 15 the

Table 5: Comparison on 100-node instances - SET 2 with $(7, 6, 5)$ vehicles

Class	ALNS			SDM		Hyb-Heur$_{10}$			Hyb-heur$_1$		GAP (%)	
	$\overline{T}(s)$	$P_{best}(\%)$	$Dev(\%)$	$T(s)$	$P(\%)$	$\overline{T}(s)$	$P_{best}(\%)$	$Dev(\%)$	$T(s)$	$P(\%)$	ALNS	SDM
$C100$	150.39	68.35	2.53	74.98	71.34	10.77	78.13	1.06	11.65	77.46	14.31	8.58
$C200$	143.92	66.58	2.65	43.03	68.23	9.23	74.54	1.36	8.79	73.54	11.95	7.79
$R100$	138.97	69.86	2.07	26.70	71.85	8.82	79.10	1.46	9.01	77.63	13.23	8.05
$R200$	144.65	71.38	2.31	21.09	73.61	8.97	81.45	1.41	8.57	80.64	14.12	9.55
$RC100$	149.33	74.70	2.64	49.90	77.27	10.34	84.89	1.15	10.41	83.81	13.64	8.46
$RC200$	146.90	74.71	2.17	48.13	79.02	10.11	85.79	1.35	9.30	84.80	14.84	7.31
$Mean$	145.69	70.93	2.39	43.97	73.55	9.71	80.65	1.30	9.62	79.65	13.68	8.29

computational times of ALNS, although they are tested on similar machines. The Hybrid-Heuristic improves also the computational times compared to SDM by a factor of almost 5.6, going from $101.19s$ to $8.98s$. Based on Tables 4 and 5, we notice that increasing the number of vehicles from $SET1$ to $SET2$ does not have a substantial impact on computational times and the deviation from the best solution of ALNS and Hybrid-Heuristic, and slight reduction in computational times of the SDM, going from $57.22s$ to $43.97s$. Regarding the objective value, all the three methods realize a substantial improvement, with an increase of 6.25% by ALNS, 7.5% by SDM and 7.8% by Hybrid-Heuristic.

Table 6: Comparison on 200-node instances - SET 1 with $(9, 8, 7)$ vehicles

Class	ALNS			SDM		Hyb-Heur$_{10}$			hyb-heur$_1$		GAP (%)	
	$\overline{T}(s)$	$P_{best}(\%)$	$Dev(\%)$	$T(s)$	$P(\%)$	$\overline{T}(s)$	$P_{best}(\%)$	$Dev(\%)$	$T(s)$	$P(\%)$	ALNS	SDM
$C100$	589.60	57.68	2.68	548.47	63.71	60.80	67.54	1.52	59.50	66.26	17.09	4
$C200$	542.64	52.60	2.93	187.14	58.89	51.73	63.16	1.29	52.16	62.47	20.07	6.08
$R100$	539.19	59.60	2.30	129.45	63.56	57.31	70.36	1.25	57.80	69.22	18.05	8.90
$R200$	542.78	59.27	2.56	154.69	62.75	58.36	70.16	1.09	57.94	69.32	18.39	10.47
$RC100$	561.80	62.18	2.06	373.47	67.10	66.54	73.84	0.96	61.30	72.90	18.76	8.65
$RC200$	570.06	62.66	1.93	349.85	67.86	60.49	74.55	1.40	69.46	73.81	18.98	8.77
$Mean$	557.68	59	2.41	290.51	63.98	59.21	69.94	1.25	59.69	69	18.56	7.81

Tables 6 and 7 report the results for 200-node instances while considering

Table 7: Comparison on 200-node instances - SET 2 with $(12, 11, 10)$ vehicles

Class	ALNS			SDM		Hyb-Heur$_{10}$			hyb-heur$_1$		GAP (%)	
	$\overline{T}(s)$	$P_{best}(\%)$	$Dev(\%)$	$T(s)$	$P(\%)$	$\overline{T}(s)$	$P_{best}(\%)$	$Dev(\%)$	$T(s)$	$P(\%)$	ALNS	SDM
$C100$	619.33	66.57	2.19	86.40	73.51	88.21	79.01	0.92	79.79	78.47	18.68	6.74
$C200$	566.36	61.46	1.81	79.50	68.78	74.93	74.77	1.04	74.19	74.14	21.67	7.79
$R100$	585.49	70.29	1.78	59.92	77.27	81.58	83.75	0.88	82.81	83.06	19.15	7.49
$R200$	589.75	70.17	2.04	77.79	76.23	80.85	83.60	0.99	85.16	82.90	19.13	8.76
$RC100$	607.04	72.46	1.72	258.62	80.30	92.83	87.05	0.94	93.90	86.48	20.13	7.70
$RC200$	633.17	73.58	1.95	175.35	80.24	84.14	87.50	0.99	89.25	86.84	18.93	8.22
$Mean$	600.19	69.09	1.92	122.93	76.05	83.76	82.61	0.96	84.18	81.98	19.62	7.79

two sets of vehicles, $(9, 8, 7)$ and $(12, 11, 10)$. In general, these results confirm the good performance of Hybrid-Heuristic demonstrated on medium-size instances by outperforming the two other methods in terms of objective value and computational times. For instance, Hybrid-Heuristic achieves an improvement gap of 18.56% and 19.62% over ALNS on $(9, 8, 7)$ vehicles and $(12, 11, 10)$ vehicles instances, respectively. Regarding SDM, our method improves the results by 7.81% and 7.79% for the same sets of instances. Hybrid-Heuristic also improves the deviation percentage on 200-node instances compared to 100-node instances, since it reduces it to 1.25% for $(9, 8, 7)$ vehicles and less than 1% on the second set ($(12, 11, 10)$ vehicles). Whereas the ALNS still has a deviation percentage around 2%.

The good performance of our method goes along with an increase in computational times, with an average of $59.21s$ for $(9, 8, 7)$ vehicles and $83.76s$ for $(12, 11, 10)$ vehicles reported from column Hyb-Heur$_{10}$, that is, an overall of $71.49s$ against $9.10s$ for 100-node instances. This is justified by the exponential explosion and the algorithm parameters, which are based on the parameters of the problem. We notice also a substantial increase in computational times of ALNS, by reaching an average of $557.68s$ and $600.19s$ for SET1 and SET2 respectively, that is, a factor of more than 4 compared to 100-node instances, and almost 8.5 times more than computational times reported by our method for 200-node instances. Regarding SDM, the computational times achieved by the method are between those of ALNS and Hybrid-Heuristic, reaching an average of $206.22s$ on 200-node instances. Interestingly, computational times of SDM remarkably decrease from $290.51s$ for $(9, 7, 6)$-vehicle instances to $122.91s$ for $(12, 11, 10)$-vehicle instances.

5. Conclusion and perspectives

In this paper, we were interested in 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 Hybrid-Heuristic method which combines a GRASP×ILS meta-heuristic and a set covering formulation as a post-optimization phase. Interestingly, the GRASP×ILS incorporates an *adaptive candidate-list* based insertion and a Variable Descent Neighborhood search heuristic. Intermediate local optima solutions found by the GRASP×ILS are temporary stored in a pool. In the post-optimization phase, a pool of routes is constructed from the pool of solutions and a set covering formulation is solved in order to improve the best solution obtained so far. A detailed computational tests prove, on the hand, the relevant of different components of the Hybrid-Heuristic, and on the other hand, the efficiency of our approach when compared to the literature. This study opens the door for several research directions. The first track is the consideration of additional criteria in the objective function such as load balancing. This criterion is relevant when we intend to equally distribute activities on protection teams and avoid them to be overwhelmed. Another promising research direction is the elaboration of a branch-and-price method based on the set covering formulation proposed in this paper.

Acknowledgements

The authors would like to thank the Hauts-de-France region and the European Regional Development Fund (ERDF) 2014/2020 for the funding of this work. This work was carried out in the framework of GEOSAFE (Geospatial Based Environment For Optimization Systems Addressing Fire Emergencies) and of Labex MS2T funded through the program "Investments for the Future" managed by the National Agency for Research (Reference ANR-11-IDEX-0004-02).

References

[1] Sohaib Afifi, Duc-Cuong Dang, and Aziz Moukrim. Heuristic solutions for the vehicle routing problem with time windows and synchronized visits. Optimization Letters, 10(3):511–525, 2016.

[2] Alfred V. Aho, Michael R Garey, and Jeffrey D. Ullman. The transitive reduction of a directed graph. SIAM Journal on Computing, 1(2):131–137, 1972.

[3] Mustafa Avci and Mualla Gonca Avci. A grasp with iterated local search for the traveling repairman problem with profits. Computers & Industrial Engineering, 113:323–332, 2017.

[4] Ilaria Baffo, Pasquale Carotenuto, and Stefania Rondine. An orienteering-based approach to manage emergency situation. Transportation research procedia, 22:297–304, 2017.

[5] Burcu Balcik. Site selection and vehicle routing for post-disaster rapid needs assessment. Transportation research part E: logistics and transportation review, 101:30–58, 2017.

[6] Burcu Balcik and İhsan Yanıkoğlu. A robust optimization approach for humanitarian needs assessment planning under travel time uncertainty. European Journal of Operational Research, 282(1):40–57, 2020.

[7] Asma Ben-Said, Racha El-Hajj, and Aziz Moukrim. A variable space search heuristic for the capacitated team orienteering problem. Journal of Heuristics, 25(2):273–303, 2019.

[8] David Bredström and Mikael Rönnqvist. Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. European journal of operational research, 191(1):19–31, 2008.

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

[10] Teodor Gabriel Crainic, Nicoletta Ricciardi, and Giovanni Storchi. Models for evaluating and planning city logistics systems. Transportation science, 43(4):432–454, 2009.

[11] Michael Drexl. Synchronization in vehicle routing—a survey of vrps with multiple synchronization constraints. Transportation Science, 46(3):297–316, 2012.

[12] Thomas A Feo and Mauricio GC Resende. Greedy randomized adaptive search procedures. Journal of global optimization, 6(2):109–133, 1995.

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

[14] Syrine Roufaida Ait Haddadene, Nacima Labadie, and Caroline Prodhon. A grasp× ils for the vehicle routing problem with time windows, synchronization and precedence constraints. Expert Systems with Applications, 66:274–294, 2016.

[15] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives, 3:43–58, 2016.

[16] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search. In Handbook of metaheuristics, pages 320–353. Springer, 2003.

[17] Julien Michallet, Christian Prins, Lionel Amodeo, Farouk Yalaoui, and Grégoire Vitry. 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, 2014.

[18] Dian Nuraiman, Melih Ozlen, and John Hearne. A spatial decomposition based math-heuristic approach to the asset protection problem. Operations Research Perspectives, 7, 2020.

[19] Sophie N Parragh and Karl F Doerner. Solving routing problems with pairwise synchronization constraints. Central European journal of operations research, 26(2):443–464, 2018.

[20] Iman Roozbeh, Melih Ozlen, and John W Hearne. An adaptive large neighbourhood search for asset protection during escaped wildfires. Computers & Operations Research, 97:125–134, 2018.

[21] Martijn Van Der Merwe, James P Minas, Melih Ozlen, and John W Hearne. A mixed integer programming approach for asset protection during escaped wildfires. Canadian Journal of forest research, 45(4):444–451, 2015.

[22] Martijn van der Merwe, Melih Ozlen, John W Hearne, and James P Minas. Dynamic rerouting of vehicles during cooperative wildfire response operations. Annals of Operations Research, 254(1-2):467–480, 2017.

[23] Fulin Xie, Chris N Potts, and Tolga Bektaş. Iterated local search for workforce scheduling and routing problems. Journal of Heuristics, 23(6):471–500, 2017.