



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

Ala-Eddine Yahiaoui, Aziz Moukrim, Mehdi Serairi

► To cite this version:

Ala-Eddine Yahiaoui, Aziz Moukrim, Mehdi Serairi. GRASP-ILS and set cover hybrid heuristic for the synchronized team orienteering problem with time windows. 2021. hal-03110595v3

HAL Id: hal-03110595

<https://hal.science/hal-03110595v3>

Preprint submitted on 21 Jul 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

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

Ala-Eddine Yahiaoui*, Aziz Moukrim and Mehdi Serairi

Université de technologie de Compiègne, CNRS, Heudiasyc (Heuristics and Diagnosis of Complex Systems), CS 60 319, 60 203 Compiègne Cedex

E-mail: ala-eddine.yahiaoui@hds.utc.fr [AE. Yahiaoui]; aziz.moukrim@hds.utc.fr [A. Moukrim]; mehdi.serairi@hds.utc.fr [M. Serairi]

Received DD MMMM YYYY; received in revised form DD MMMM YYYY; accepted DD MMMM YYYY

Abstract

Wildfires are a natural phenomenon that regularly occurs 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 and human casualties. Several initiatives have been undertaken in the last years in order to apply operations research tools to help firefighting teams schedule and optimize their protection activities when dealing with wildfires.

In this context, a recent variant of the Team Orienteering Problem, referred to as the Asset Protection Problem, was proposed in [30]. In this problem, firefighting teams provide a protective service to a set of assets endangered by wildfires. These activities can be performed by a heterogeneous fleet of vehicles and occur within specific time intervals estimated on the basis of fire fronts progression. This problem incorporates three additional constraints: *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 based on a set covering formulation. Interestingly, GRASP×ILS incorporates an *adaptive candidate list-based insertion* heuristic and a Variable Neighborhood Descent search procedure. Detailed computational tests were carried out on benchmark instances from the literature. The results show that our method outperforms the other methods in the literature, since it improves all the best known solutions on medium- and large-size instances, while maintaining shorter computational times.

Keywords: Team Orienteering Problem · Synchronization · Greedy Randomized Adaptive Search Procedure · Set Covering

* Author to whom all correspondence should be addressed (e-mail: ala-eddine.yahiaoui@hds.utc.fr).

1. Introduction

In the last decade, wildfires have become a frequent phenomenon, causing major damage to private property, community assets and human life. Many countries have recently witnessed the devastating impact of wildfires on nature and human activities: Portugal (2017), Sweden (2018), Australia (2019-2020), California (2020). Asset protection activities performed by Incident Management Teams (IMT) during wildfires are therefore of crucial importance in order to minimize the risk of losing vital infrastructures. However, several challenging tasks and difficulties complicate the working environment of IMT within which critical and complex decisions must be made. Consequently, the application of operations research methods, either exact or heuristic, can reinforce the management of wildfires in such hostile situations.

In this context, Van Der Merwe et al. [30] consider the case of an out-of-control wildfire spreading across a landscape and threatening a number of assets like bridges, electrical substations, schools and factories. Defensive activities carried out by IMT near assets before fire impact are important to reduce the risk of losing them. Examples of defensive tasks include the removal of debris and combustible materials, wetting down buildings and putting out fires. Moreover, IMT activities should not be implemented too early, or the intervention will be ineffective. For this purpose, fire progression can be estimated 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. [30] 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 considered 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 resource requirements that are expressed by the number and type of required vehicles. Due to these constraints, protecting all the assets might be impossible. Hence, a value, referred to as 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 begin 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 refer to 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 [13]. Extensive researches have been conducted during the last decade to study the TOP and its variants. Several solution approaches have been proposed. As exact approaches, Boussier et al. [11] introduced a column generation based approach for the TOP, whereas El Hajj et al. [17] proposed a cutting plane approach reinforced by dominance properties and symmetry cuts. For the same problem, Bianchessi et al. [10] presented a branch-and-cut method. Regarding other variants of the TOP, Archetti et al. [3] designed a generic column generation method for two problems: the Capacitated TOP (CTOP) and the Capacitated Profitable Tour Problem (CPTP), while Yahiaoui et al. [34] proposed a cutting plane method coupled with pre-processing procedures for the Clustered TOP. Several heuristic approaches have also been proposed for the TOP and its variants. The first method for the TOP was proposed by Chao et al. [13]. Archetti et al.

[4] proposed a Tabu Search (TS) and two versions of Variable Neighborhood search (VNS). Dang et al. [15] introduced a novel splitting procedure for giant tours embedded inside a PSO algorithm. Archetti et al. [3] proposed two versions of TS and VNS for the CTOP, while Ben-Said et al. [9] proposed a GRASP \times ELS approach to the same problem. We refer the reader to [20] for a detailed survey on the TOP.

Several variants of the TOP have been introduced in order to solve different issues related to disaster management in general. Baffo et al. [6] 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 [7] 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, can therefore 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 the 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 [8]. 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 [12] studied a variant of VRP with synchronized visits in the context of home-care services. The same problem was addressed by Parragh et al. [27] and used to solve real problems related to technicians routing and scheduling problems. Crainic et al. [14] presented a VRP variant for city logistics that includes, in addition to two-echelon network architecture and multi-trip routes, exact synchronization constraints between vehicles of first and second echelons at the level of satellites. We refer to Drexl [16] for a detailed survey on synchronization in vehicle routing problems.

Regarding the resolution approaches for the STOPTW, Van Der Merwe et al. [30] introduced a mixed integer programming model for the problem, which was demonstrated on a realistic wildfire scenario in Tasmania. The authors in [29] proposed an adaptive large neighborhood search heuristic (ALNS) for the problem along with a new set of benchmark instances. A spatial decomposition math-heuristic (SDM) resolution approach was recently proposed by Nuraiman et al. [26] for the same problem. This method succeeded in substantially improving the best solution for all medium-size and large-size instances compared with [29]. In the same vein, the authors in [31] 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 developed earlier.

In this paper, we propose the Hybrid Heuristic method composed of a Greedy Randomized Adaptive Search Procedure (GRASP) coupled with an Iterated Local Search (ILS) and a set covering formulation. GRASP \times 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 \times ILS. Our method has several similarities with the iterative three-component heuristic (I3CH) proposed in [22] for the TOPTW. However, the main difference between the two approaches is that in our case, the set covering problem is solved once at the end of the algorithm, whereas in I3CH, the set cover is executed at the end of each iteration. This difference is mo-

tivated by the fact that our set covering formulation includes additional side constraints that considerably increase the computational time burden if it is launched at each iteration of the GRASP×ILS.

The remainder of this paper is organized as follows. The mathematical formulation of the STOPTW is presented 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 $V = \{0, 1, \dots, n+1\}$ is the set of vertices and $A = \{(i, j) : i, j \in V, i \neq j\}$ is the set of arcs. $V^- = \{1, \dots, n\}$ represents the set of assets whereas vertices 0 and $n+1$ represent the departure and the arrival depots, respectively. For convenience, we define two other index sets, $V^d = \{0, 1, \dots, n\}$ and $V^a = \{1, \dots, n, n+1\}$. A 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 $r_i = \langle r_{i1}, r_{i2}, \dots, r_{i|Q|} \rangle$, where r_{iq} is the number of vehicles per type $q \in Q$ required by asset i .
- service duration a_i , which is the time needed to protect asset i .
- profit p_i that represents the value of asset i .

The decision variables used in the mathematical formulation are:

- y_i : binary decision variable, which takes the value of 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 $q \in Q$; 0 otherwise.
- x_{ijq} : integer decision variable, which indicates the number of vehicles of type $q \in Q$ traversing the arc $(i, j) \in A$.

The mathematical formulation of the STOPTW is as follows :

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

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

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

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

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

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

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

$$o_i \leq s_i \leq c_i \quad \forall i \in V^- \quad (8)$$

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

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

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

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 are 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 window 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 an NP-hard problem. This formulation only succeeds in solving small-size instances of the problem. Using the com-

mercial solver Cplex, instances with at most 34 assets have been successfully solved in [29]. Due to this observation, we propose the Hybrid Heuristic to tackle medium- and large-size instances below.

3. Solution approach

The Greedy Randomized Adaptive Search Procedure (GRASP) is a multi-start local search metaheuristic introduced by Feo and Resende in [18]. 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 [24]. The basic idea behind this method is to construct a new solution in each iteration 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 the 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 [33], the Traveling Repairman Problem with Profits [5], the Periodic VRP with Time Windows [25], and the VRP with Synchronization and Precedence Constraints [21]. 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 GRASP and always performed after the ILS. In our case, the ILS incorporates 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 (see Section 3.3).

3.1. General flow

The GRASP×ILS metaheuristic is outlined 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}_{pool} (lines 22-23). A post-optimization phase consists in constructing a pool of routes \mathcal{P} from \mathcal{S}_{pool} (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 do that, the algorithm randomly selects a number of assets (comprised between 1 and d_{max}) in each iteration of the ILS and removes them 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 allows the ILS to escape from local optima.

Algorithm 1: HYBRID HEURISTIC ALGORITHM

```

1  $S_{best} \leftarrow \emptyset$ 
2 for ( $i = 1; i \leq IterMax_G; i++$ ) do
3    $(\alpha, \beta, \gamma) \leftarrow (0.5, 0.5, 0.5)$ 
4    $S \leftarrow \emptyset$ 
5    $(S, \alpha, \beta, \gamma) \leftarrow AdaptiveInsertion(S, \alpha, \beta, \gamma)$  (see Section 3.2)
6    $S_{ILS} \leftarrow S$ 
7    $\mathcal{S}_{pool} \leftarrow \mathcal{S}_{pool} \cup \{S\}$ 
8    $Iter \leftarrow 0$ 
9    $d_{max} \leftarrow 3$ 
10  while  $Iter < IterMax$  do
11     $d \leftarrow \mathcal{U}(1, d_{max})$ 
12    Remove  $d$  assets from  $S$ 
13    Update  $S$ 
14     $(S, \alpha, \beta, \gamma) \leftarrow AdaptiveInsertion(S, \alpha, \beta, \gamma)$  (see Section 3.2)
15    if ( $Profit(S) > Profit(S_{ILS})$ ) then
16       $S_{ILS} \leftarrow S$ 
17       $d_{max} \leftarrow 3$ 
18       $Iter \leftarrow 0$ 
19    else
20       $d_{max} \leftarrow d_{max} + 1$ 
21       $Iter \leftarrow Iter + 1$ 
22   $S_{VND} \leftarrow VariableNeighborhoodDescent(S_{ILS})$  (see Section 3.3)
23   $\mathcal{S}_{pool} \leftarrow \mathcal{S}_{pool} \cup \{S_{VND}\}$ 
24  if ( $Profit(S_{VND}) > Profit(S_{best})$ ) then
25     $S_{best} \leftarrow S_{VND}$ 
26   $\mathcal{P} \leftarrow InitPool(\mathcal{S}_{pool})$  (see Section 3.4)
27   $S_c \leftarrow setCover(\mathcal{P})$  (see Section 3.4)
28  if ( $Profit(S_c) > Profit(S_{best})$ ) then
29     $S_{best} \leftarrow S_c$ 
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 then adds 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. This criterion takes the following factors into consideration:

- *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 probably 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. It is therefore 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. We therefore 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 do this, 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.

The insertion criterion is therefore calculated for each asset as follows:

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

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, allow the insertion heuristic to cover a large part of the search space [9]. 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. Six functions $f_l, l \in \{1, \dots, 6\}$ are then used to determine six new combinations of (α, β, γ) at each iteration. In the first four functions $f_l, l \in \{1, \dots, 4\}$, 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 favor the insertion of assets with higher profits, since the aim of solving the STOPTH is to maximize the total collected profit. However, we consider a last function where all the parameters are randomly generated 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 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.

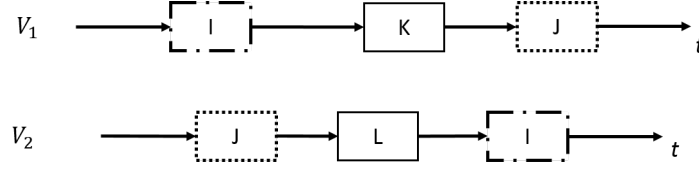


Figure 1: Cross-synchronization situation

3.2.1. Preventing cross-synchronization

A key characteristic in the STOPTW is that the routes are interdependent and, consequently, 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 of exactly the same types, a situation of cross-synchronization occurs when a first vehicle visits asset j before i , while a second vehicle visits asset i before j . An example of a cross-synchronization situation is illustrated by Figure 1. To avoid such situations, we proceed in a way similar to that of Afifi et al. [1].

Let us consider an auxiliary graph that we refer to as 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 link between any successive visits in S , i.e. if asset j is visited immediately after asset $i \in V^-$ in a given route $R_k \in R$, a precedence relation (an arc) from asset i to asset j is added to X . The precedence relations are then propagated beyond immediate successors using the transitive closure [2].

The construction of the precedence graph from scratch is performed by launching a depth-first search from each node in V^- , yielding a time complexity of $\mathcal{O}(n(n + E))$, where E is an upper bound on the number of arcs in H , with a worst case value of less than $\frac{n(n-1)}{2}$. As a result, for each asset $i \in V^-$, we obtain a set $\Gamma^+(i)$ that defines the set of successors in H .

Hence, in order to check the feasibility of the insertion of a visit of a given asset x between two visits of assets i and j in an arbitrary route $R_k \in R$, it is sufficient to verify that $i \notin \Gamma^+(x)$ and $x \notin \Gamma^+(j)$. It should be mentioned that this test is only performed before the insertion of visits of assets that require multiple vehicles.

Once the transitive closure of H is completed along with the successors lists $\Gamma^+(i), i \in V^-$, a matrix representation Ψ is constructed in $\mathcal{O}(n^2)$ to allow a constant time access to precedence relations. A precedence relation from asset i to asset j is simply represented in Ψ by setting $\Psi[i][j] = 1$. As a result, using Ψ , the feasibility test is performed in a constant time.

The construction of the transitive closure of H and its matrix representation Ψ from scratch is performed after each 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 $x \in V^-$ in a given route after a visit of asset i and before a visit of asset j . The update of Ψ is as follows. The values of $\Psi[i][x]$ and $\Psi[x][j]$ are immediately set to 1 since i precedes x and x precedes j . Any asset $h \in V^-$, that is preceded by j , must now be preceded by x . Thus, if $\Psi[j][h] = 1$, we set the value of $\Psi[x][h]$ to 1. Similarly, any asset $l \in V^-$, that precedes i , must now precede x . Thus, if $\Psi[l][i] = 1$, then

$\Psi[l][x]$ is also set to 1. Now, for any asset $l \in V^-$ that precedes x , and any asset $h \in V^-$ that is preceded by j , l must now precede h . As a result, if $\Psi[l][x] = 1$ and $\Psi[j][h] = 1$, the value of $\Psi[l][h]$ is set to 1. Similarly, for any asset $l \in V^-$ that precedes i , and any for asset $h \in V^-$ that is preceded by x , l must now precede h . As a result, if $\Psi[l][i] = 1$ and $\Psi[x][h] = 1$, the value of $\Psi[l][h]$ is set to 1.

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

3.2.2. Time-window feasibility check

Time window 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 Vansteenwegen et al. [32] for the TOPTW. In Afifi et al. [1], the authors propose a generalization of that approach to include variants of VRPTW with synchronization. The authors propose to record some useful information in data structures and update them after each insertion or partial destruction. An assumption is made that no cross-synchronization situations exist in the current solution S .

Before proceeding further, let us first define a feasible solution S as a set of routes $R = (R_1, R_2, \dots, R_{|R|})$ performed by distinct vehicles. Each route $R_k \in R$ is an ordered list of visits starting from the departure depot and ending at the arrival depot. We denote the p^{th} asset visited in route R_k by $R_k(p)$. We also denote the set of routes that visit the asset $i \in V^-$ by $R^i \subseteq R$.

Following are useful notations:

- 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 of 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 follows that:

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

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

$$Wait_i^k = Str_i - Arr_i^k, \quad k : R_k \in R^i \quad (14)$$

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

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

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

$$Maxshift_i = \min_{k: R_k \in R^i} \{Maxshift_i^k\} \quad (16)$$

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

$$Shift_x^{k,p+1} = t_{R_k(p)x} + Wait_x^k + a_x + t_{x,R_k(p+1)} - t_{R_k(p),R_k(p+1)} \quad (17)$$

Hence, the insertion of a visit of an asset x between positions p and $p + 1$ is feasible in terms of time windows if:

$$Str_{R_k(p)} + a_{R_k(p)} + t_{R_k(p)x} \leq c_x \quad \text{and} \quad Shift_x^{k,p+1} \leq Wait_{R_k(p+1)}^k + Maxshift_{R_k(p+1)}. \quad (18)$$

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 two phases: a forward and a backward pass. The forward pass is performed to compute the earliest starting times Str_i at each node, whereas the backward pass is used to calculate the maximum delays $MaxShift_i$. Hence, the complexity of updating this information is $\mathcal{O}(n + E)$, while the time-window feasibility test is performed in a constant time.

3.2.3. Candidate list-based insertion algorithm

The *candidate list-based insertion* is described in Algorithm 2. It starts by sorting the unrouted assets in non-decreasing order of the criterion presented by the Equation 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 vehicles, a list of potential insertion positions are computed and stored in Δ (line 9). The algorithm then 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 is the *time-windows* feasibility check (line 16). If the position does not satisfy one of the feasibility tests, it is 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 the 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 the requirements of a given asset are not fully satisfied at the end, 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 to recompute service start times and maximum delays as well. The whole process of insertion of assets is reiterated until all the candidates in σ are tested.

Algorithm 2: LIST-BASED INSERTION

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

```
1  $\sigma \leftarrow$  Sort unrouted assets to non-decreasing values of  $Cr_i(\alpha, \beta, \gamma)$  (See formula (12))
2  $insert \leftarrow True$ 
3 while  $insert$  and  $\sigma \neq \emptyset$  do
4    $insert \leftarrow False$ 
5   for  $i = 1$  to  $|\sigma|$  do
6      $k_{max} \leftarrow |r_{\sigma_i}|$ 
7      $k \leftarrow 0$ 
8     for  $q = 1$  to  $|Q|$  do
9        $\Delta \leftarrow$  list of all positions in routes of type  $q \in Q$  of  $S$ 
10       $cpt \leftarrow 1$ 
11      while  $cpt \leq r_{\sigma_i, q}$  do
12         $foundPos \leftarrow False$ 
13        while  $foundPos = False$  and  $\Delta \neq \emptyset$  do
14          Select a random position  $(q, r, p) \in \Delta$  //  $r$ : a route of  $S$  of type
             $q \in Q$ ,  $p$ : position in  $r$ 
15          if  $k = 0$  or  $(q, r, p)$  is cross-synchro-feasible for  $\sigma_i$  then
16            if  $(q, r, p)$  is time-window-feasible for  $\sigma_i$  then
17              Insert  $\sigma_i$  in position  $(q, r, p)$ 
18              Update  $S$ 
19               $foundPos \leftarrow True$ 
20               $cpt \leftarrow cpt + 1$ 
21               $k \leftarrow k + 1$ 
22             $\Delta \leftarrow \Delta \setminus \{(q, r, p)\}$ 
23          if  $foundPos = False$  then
24             $break$  // no feasible position in  $\Delta$ 
25        if  $cpt \leq r_{\sigma_i, q}$  then
26           $break$  // requirements of type  $q$  are not fully satisfied
27      if  $(k < k_{max})$  then
28        Remove all visits of  $\sigma_i$  from  $S$ 
29        Update  $S$ 
30      else  $insert \leftarrow True$ 
31 return  $S$ 
```

3.3. Variable Neighborhood Descent

The *adaptive candidate list-based insertion* operator and its sorting criterion allow our method to cover a large part of the search space. However, the sorting criterion overlook a key performance parameter,

which is the **travel cost** induced by visit insertions. This choice is justified by computational time burden when constructing the candidate list, since the insertion cost depends on the insertion position, which is *a priori* unknown. On the contrary, the profit, the width of time windows and the resource requirements are all known beforehand. The Variable Neighborhood Descent (VND) described hereafter is precisely proposed in order to cope with this drawback. **It is noteworthy to mention that we do not consider waiting times when evaluating local search moves. We do only consider arc costs derived from the distance matrix.**

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

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

If the two first operators succeed in reducing the **travel costs** of the solution, this may create room to insert profitable assets into the solution, and consequently, improving the total profit. The second set of 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 efficiency, the previously described operators (*relocate* and *replacement*) are implemented with a reduced-size neighborhood. Due to the synchronization constraints, checking the feasibility of some moves in the original version of these operators requires first removing and then inserting subsets of assets. This yields a $\mathcal{O}(n^2)$ 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 in a constant time. Regarding the *replacement* operator, the reduced neighborhood is described as follows. Let us consider the replacement of asset u by an unrouted asset v . Such a move is considered only if $p_v > p_u$ and $r_{uq} \geq r_{vq} \quad \forall q \in Q$. For the *relocate* operator, we only consider assets with one required visit and relocation to a different route other than the current one. It is also noteworthy to mention that for the neighborhoods *2-opt** and *candidate list-based insertion*, a feasibility check is already performed in $\mathcal{O}(1)$ for all the moves.

Algorithm 3 describes the VND in detail. The set $N^l = \{N_1^l, N_2^l\}$ contains *replacement* (N_1^l) and *relocate* (N_2^l) operators, whereas the set $N^p = \{N_1^p, N_2^p\}$ contains *2-opt** (N_1^p) and *candidate list-based insertion* (N_2^p) operators. The first level of VND (lines 6-13) tries to improve the **travel costs**, and the second level (lines 14-22) focuses on improving the profit. Neighborhoods are sequentially applied one-by-one at each level. Each level of VND is iteratively executed until no neighborhood succeeds in improving the solution. In addition, if the second level neighborhoods succeed in improving the profit, the whole VND starts a new iteration, otherwise, the algorithm is terminated.

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 being sub-optimal, may incorporate some good individual routes. Moreover, individual routes may happen to be promising

Algorithm 3: VARIABLE NEIGHBORHOOD DESCENT

Input: Solution S

```
1  $k_{max}^1 \leftarrow |N^l|$ 
2  $k_{max}^2 \leftarrow |N^p|$ 
3  $impr \leftarrow True$ 
4 while  $impr = True$  do
5    $impr \leftarrow False$ 
6    $imprTravelTime \leftarrow True$ 
7   while  $imprTravelTime = True$  do
8      $imprTravelTime \leftarrow False$ 
9     for  $k^1 = 1$  to  $k_{max}^1$  do
10       $S' \leftarrow N_{k^1}^l(S)$ 
11      if  $TravelTime(S') < TravelTime(S)$  then
12         $S \leftarrow S'$ 
13         $imprTravelTime \leftarrow True$ 
14    $imprProfit \leftarrow True$ 
15   while  $imprProfit = True$  do
16      $imprProfit \leftarrow False$ 
17     for  $k^2 = 1$  to  $k_{max}^2$  do
18        $S' \leftarrow N_{k^2}^p(S)$ 
19       if  $Profit(S') > Profit(S)$  then
20          $S \leftarrow S'$ 
21          $imprProfit \leftarrow True$ 
22          $impr \leftarrow True$ 
23 return  $S$ 
```

if combined with routes from other solutions and can yield new improved solutions. For that purpose, we propose a route recombination procedure as a post-optimization phase in order to obtain the best feasible solution.

During the search process, solutions produced by GRASP \times ILS combined with VND are stored in a set \mathcal{S}_{pool} . At the end, single routes are extracted from \mathcal{S}_{pool} and saved in a set of pools $\{\mathcal{T}^q | q \in Q\}$, where $\mathcal{T}^q = \{T_1^q, T_2^q, \dots, T_{|\mathcal{T}^q|}^q\}$. The 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 us first introduce some necessary notations and define the decision variables. In addition to the variables y and s introduced in Section 2, we use a binary decision variable θ_k^q , $q \in Q$, $1 \leq k \leq |\mathcal{T}^q|$ to indicate whether a route T_k^q is selected or not in the solution found by the solver.

We denote the asset immediately visited after the node $i \in V^d$ in route T_k^q by $\gamma_{qk}^+(i) = \{j\}, j \in V^-$. In the case where node i is the last visited node, or it is not visited at all in T_k^q , $\gamma_{qk}^+(i)$ is set to empty set.

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

$$A^q = (a_{ik}^q) \text{ with } a_{ik}^q = \begin{cases} 1 & \text{if asset } i \in T_k^q \\ 0 & \text{otherwise} \end{cases}$$

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

$$[SCP1] \quad \max \sum_{i \in V^-} p_i y_i \quad (19)$$

$$\sum_{k: T_k^q \in \mathcal{T}^q} a_{ik}^q \theta_k^q \geq r_{iq} y_i \quad \forall q \in Q, \forall i \in V^- \quad (20)$$

$$\sum_{k: T_k^q \in \mathcal{T}^q} \theta_k^q \leq P_q \quad \forall q \in Q \quad (21)$$

$$s_i + t_{ijq} + a_i - s_j \leq M(1 - \theta_k^q) \quad \forall q \in Q, \forall k: T_k^q \in \mathcal{T}^q, \forall i \in T_k^q, j \in \gamma_{qk}^+(i) \quad (22)$$

$$t_{0iq} - s_i \leq M(1 - \theta_k^q) \quad \forall q \in Q, \forall k: T_k^q \in \mathcal{T}^q, i \in \gamma_{qk}^+(0) \quad (23)$$

$$o_i \leq s_i \leq c_i \quad \forall i \in V^- \quad (24)$$

$$y_i \in \{0, 1\} \quad \forall i \in V^- \quad (25)$$

$$\theta_k^q \in \{0, 1\} \quad \forall q \in Q, \forall k, 1 \leq k \leq |\mathcal{T}^q| \quad (26)$$

[SCP1] aims at maximizing the total collected profit (19) subject to the set of constraints (20-24). Constraints (20) ensure the satisfaction of resource requirements for assets, whereas constraints (21) impose an upper limit on the number of vehicles.

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 introduce constraints (22) and (23) in which only one service starting time decision variable s_i is used per asset i . In this way, we impose the condition that the same

asset i must be visited in the selected routes at the same time. Moreover, a minimum travel time is imposed between any two consecutive visits in selected routes. Constraints (22) and (23) are systematically deactivated if the route is not selected. Finally, constraints (25-26) are domain definitions.

It is noteworthy to mention that in this formulation, due to the inequality used in constraints (20), it is possible to visit assets a number of times more than required, or even visit assets that do not belong to the solution found by [SCP1]. The choice of a set covering-based formulation instead of a set partitioning based on is motivated by the fact that the latter would be too restrictive and may miss some good solutions. Indeed, the solution found must undergo a reparation phase in order to get rid of the visits of non-selected assets as well as extra visits from the selected routes.

Another important drawback raised by this formulation is that travel time constraints should be systematically respected between any successive visits in the selected routes, including the visits related to assets that are not selected in the final solution. This inconvenience tightens travel time constraints and potentially prevents the solver from extracting promising solutions from the pools of routes.

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

$$[SCP2] \quad (19), (20), (21), (24), (25), (26)$$

$$s_i + t_{ijq} + a_i - s_j \leq M(3 - y_i - y_j - \theta_k^q) \quad \forall q \in Q, \forall k : T_k^q \in \mathcal{T}^q, \forall i \in V^-, j \in \Gamma_{qk}^+(i) \quad (27)$$

$$t_{0iq} - s_i \leq M(2 - y_i - \theta_k^q) \quad \forall q \in Q, \forall k : T_k^q \in \mathcal{T}^q, \forall i \in \Gamma_{qk}^+(0) \quad (28)$$

According to [SCP2], a minimum travel time is imposed between any two consecutive visits of selected assets in the selected routes. Constraints (27) and (28) 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]. Nevertheless, the solution found by [SCP2] requires a cleaning phase in order to get rid of unnecessary visits in the case where a given asset is visited often than required.

The pool size is a critical performance parameter. Adding all the feasible routes to the pool may incur a long computational time. In our case, we impose 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 given 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 of vehicles. In this case, these routes are all inserted into the pool. In this way, we ensure that the set covering formulation does not miss the best solution obtained by GRASP×ILS so far. We also propose to accelerate our method by solving the LP relaxation of the integer programming model. If the relaxation is no better than the best solution

already found, the resolution of the integer programming model 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 closely dependent on the quality of the routes, which are exclusively generated by GRASP×ILS and VND.

4. Computational tests

In this section, we investigate the performance of the Hybrid Heuristic method. It was implemented using C++ and the 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 [29] to evaluate STOPTH methods were generated based on 60 instances initially proposed for the VRPTH in [19]. 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 a combination of both (RC). Each class pattern is divided into two 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, where each one 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: $SET1 = (6, 5, 4)$ and $SET2 = (7, 6, 5)$ for 100-node instances; and $SET1 = (9, 8, 7)$ and $SET2 = (12, 11, 10)$ for 200-node instances. As a result, the benchmark is composed of 240 instances, with 120 instances for each size (100 and 200).

4.2. Parameter settings

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, *iRace* software, introduced by Lopez-Ibanez et al. [23]. We selected an arbitrary set of 24 benchmark instances and gave *iRace* a tuning budget of 2000 experiments. Four parameters are considered for tuning: the number of iterations of 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 $n + \lambda \bar{q}$, where \bar{q} is the

average number of vehicles per type calculated as $\bar{q} = \frac{\sum_{q \in Q} P_q}{|Q|}$, and λ is a weight parameter determining the influence of \bar{q} on the value of $IterMax$. Regarding GRASP, which is a set of independent iterations of the ILS, the stopping criterion is set to the average number of vehicles per type $IterMax_G = \bar{q}/\mu$, where μ is determined after tuning. Regarding the set covering problem formulation, two parameters were considered: the size of the pool (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 the *irace* package, we suggest the parameter settings depicted in Table 1.

Table 1: Parameter settings for the Hybrid Heuristic

Parameter	λ	μ	$ S $	ζ
Value	0.25	0.5	200	0.035

4.3. Sensitivity analysis

In this section, we present 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 GRASP×ILS. Different configurations were considered and run a single time on all the benchmark instances. To clearly highlight the outcome of each component in this paper, we 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 the sum of the total collected profit and the sum of computational times, respectively.

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, either in terms of objective value or computational time. Regarding the objective value, SCP2 outperforms SCP1. This is justified by the fact that SCP1 systematically imposes all the visits in the selected route of the optimal solution, which substantially reduces the solution space, and hence, potentially missing solutions with better objective value. Interestingly, SCP2 also achieved better computational times than SCP1, with a total time of 8983.55 s compared to 11655.82 s.

4.3.2. Analysis of components

As a result of the calculations made in the previous section, the second formulation [SCP2] was chosen to be part of the Hybrid Heuristic. We now investigate 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 VND and the post-optimization

Table 2: Comparison between the two set covering formulations

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

phase, denoted by SCP2 in Table 3. We begin with the post-optimization phase in the second column, which improves the aggregated profit by roughly 1089 units compared to GRASP×ILS, equivalent to an improvement of 0.23%. This improvement comes at the expense of a substantial increase in computational time, reaching 8983.55 s compared 2374.96 s for GRASP×ILS. Interestingly, we notice in the fourth column that the VND leads to a substantial improvement of the results, with a gap of 2.06% compared to GRASP×ILS, along with a reasonable increase in computational time, of about 4500 s, roughly half of the computational time of GRASP×ILS + SCP2. The reason for this is that the SCP2 includes time-related side constraints that destroy the original set covering structure. As a result, MIP solvers struggle to solve the problem, even on a small number of routes. Finally, the combination of the three components, depicted in the last column, leads to the best results in terms of objective values, reaching an improvement of 2.12% compared to GRASP×ILS. Regarding computational time, the three-component configuration is better than GRASP×ILS+SCP2. This is mainly due to the fact that the substantial improvement yielded by the VND 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+VND.

Table 3: Sensitivity analysis of the three components

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

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 to two methods from the literature, namely, an ALNS method proposed in Roozbeh et al. [29] and a mathheuristic approach known as the SDM approach, proposed in Nuraiman [26]. To evaluate our method, we proceed in a way similar to the one in [29]. We run our algorithm ten times on each instance and we record the average computational time, the best objective value as well as the average objective value of the ten runs. Since it is the same protocol, the comparison between these two methods is straightforward. On the other hand, the SDM approach as described in [26] is a deterministic method and is therefore executed only once for each instance. Consequently, in order to guarantee a fair comparison between the Hybrid Heuristic and SDM approach, we compare the objective value obtained by the first run of our method to the objective value of the SDM approach. **It is noteworthy to mention that the algorithm of ALNS presented in [29] has been improved and new results for instances with 100 nodes has been published in [28].**

Tables 4-7 show a comparison between the three methods. The results are grouped according to the sub-classes present in the benchmark instances. In Tables 4-7, we denote the results of the first run of our method by Hyb-Heur₁, whereas the results of the ten runs are denoted by Hyb-Heur₁₀. We provide the following performance measurements for the different methods:

- $P(\%)$: the percentage of profit related to the protected assets achieved by SDM and Hyb-Heur₁.
- $P_{best}(\%)$: the best percentage of profit related to the protected assets achieved by ALNS or Hyb-Heur₁₀.
- $T(s)$: computational times achieved by SDM and Hyb-Heur₁.
- $\bar{T}(s)$: average computational times achieved by ALNS or Hyb-Heur₁₀.
- $Dev(\%)$: the deviation of the average results from the best results achieved by ALNS or Hyb-Heur₁₀. It is calculated as follows:

$$Dev = \frac{P_{best} - \bar{P}}{P_{best}} \quad (29)$$

where \bar{P} is the average percentage of profit related to the protected assets achieved by ALNS or Hyb-Heur₁₀.

- $Gap(\%)$: 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 calculate the gap between the Hybrid Heuristic and ALNS is :

$$GAP = \frac{P_{best}^{Hyb-Heur_{10}} - P_{best}^{ALNS}}{P_{best}^{ALNS}}. \quad (30)$$

The gap between the Hybrid Heuristic and SDM is :

$$GAP = \frac{P_{best}^{Hyb-Heur_1} - P^{SDM}}{P^{SDM}}. \quad (31)$$

A positive value of the *GAP* means that the 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 (%)	
	$\bar{T}(s)$	$P_{best}(\%)$	$Dev(\%)$	$T(s)$	$P(\%)$	$\bar{T}(s)$	$P_{best}(\%)$	$Dev(\%)$	$T(s)$	$P(\%)$	ALNS	SDM
<i>C100</i>	135.83	63.18	1.33	95.45	65.09	9.21	70.20	1.02	8.42	69.19	11.11	6.29
<i>C200</i>	130.64	61.92	1.39	44.48	61.17	7.55	67.38	1.56	7.01	66.55	10.96	8.79
<i>R100</i>	125.48	64.49	1.71	44.30	63.68	7.74	70.49	1.47	7.50	69.86	9.30	9.69
<i>R200</i>	129.17	66.75	1.42	16.35	65.88	8.31	73.29	1.34	8.15	72.27	9.80	9.69
<i>RC100</i>	146.12	69.98	1.63	63.51	69.57	9.36	77.68	1.49	9.65	76.29	11	9.66
<i>RC200</i>	130.35	70.45	1.56	79.26	70.88	8.83	78.02	1.35	9.29	76.81	10.75	8.37
<i>Mean</i>	132.93	66.13	1.50	57.22	66.05	8.50	72.84	1.37	8.34	71.83	10.15	8.75

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

Class	ALNS			SDM		Hyb-Heur ₁₀			Hyb-heur ₁		GAP (%)	
	$\bar{T}(s)$	$P_{best}(\%)$	$Dev(\%)$	$T(s)$	$P(\%)$	$\bar{T}(s)$	$P_{best}(\%)$	$Dev(\%)$	$T(s)$	$P(\%)$	ALNS	SDM
<i>C100</i>	154.83	69.70	1.78	74.98	71.34	10.77	78.13	1.06	11.65	77.46	12.09	8.58
<i>C200</i>	136.67	67.73	1.48	43.03	68.23	9.23	74.54	1.36	8.79	73.54	10.05	7.79
<i>R100</i>	131.66	71.31	1.47	26.70	71.85	8.82	79.10	1.46	9.01	77.63	10.92	8.05
<i>R200</i>	134.30	72.74	1.48	21.09	73.61	8.97	81.45	1.41	8.57	80.64	11.97	9.55
<i>RC100</i>	152.94	75.85	1.20	49.90	77.27	10.34	84.89	1.15	10.41	83.81	11.92	8.46
<i>RC200</i>	139.50	75.79	1.11	48.13	79.02	10.11	85.79	1.35	9.30	84.80	13.19	7.31
<i>Mean</i>	141.65	72.19	1.42	43.97	73.55	9.71	80.65	1.30	9.62	79.65	11.72	8.29

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 both in terms of objective value and computational time. For instance, the Hybrid Heuristic achieves an improvement gap of 10.15% compared to ALNS for (6, 5, 4) vehicles, and a gap of 11.72% for (7, 6, 5) vehicles. Compared to the SDM approach, the 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 ALNS in terms of solution quality when executed

several times, with a deviation from the best solution of no more than 1.37% for *SET1* and 1.3% for *SET2*, compared to 1.50% and 1.42% for ALNS. The Hybrid Heuristic also achieves the best computational time, since it divides the computational times of ALNS by a factor of more than 15, although they are tested on similar machines. The Hybrid Heuristic also improves computational time compared to the SDM by a factor of almost 5.6, going from 101.19 s to 8.98 s. Based on Tables 4 and 5, we can observe that increasing the number of vehicles from *SET1* to *SET2* does not have a substantial impact on computational time or the deviation from the best solution of ALNS and the Hybrid Heuristic, and leads to a slight reduction in computational time of the SDM, decreasing from 57.22 s to 43.97 s. Regarding the objective value, all three methods show a substantial improvement compared to the first set of vehicles, with an increase of 6.06% by ALNS, 7.5% by SDM and 7.8% by the 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 (%)	
	$\bar{T}(s)$	$P_{best}(\%)$	$Dev(\%)$	$T(s)$	$P(\%)$	$\bar{T}(s)$	$P_{best}(\%)$	$Dev(\%)$	$T(s)$	$P(\%)$	ALNS	SDM
<i>C100</i>	589.60	57.68	2.68	548.47	63.71	60.80	67.54	1.52	59.50	66.26	17.09	4
<i>C200</i>	542.64	52.60	2.93	187.14	58.89	51.73	63.16	1.29	52.16	62.47	20.07	6.08
<i>R100</i>	539.19	59.60	2.30	129.45	63.56	57.31	70.36	1.25	57.80	69.22	18.05	8.90
<i>R200</i>	542.78	59.27	2.56	154.69	62.75	58.36	70.16	1.09	57.94	69.32	18.39	10.47
<i>RC100</i>	561.80	62.18	2.06	373.47	67.10	66.54	73.84	0.96	61.30	72.90	18.76	8.65
<i>RC200</i>	570.06	62.66	1.93	349.85	67.86	60.49	74.55	1.40	69.46	73.81	18.98	8.77
<i>Mean</i>	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 two sets of vehicles, (9, 8, 7) and (12, 11, 10). In general, these results confirm the good performance of the Hybrid Heuristic demonstrated on medium-size instances by outperforming the two other methods in terms of objective value and computational time. For instance, the Hybrid Heuristic achieves an improvement gap of 18.56% and 19.62% over ALNS on (9, 8, 7)-vehicle and (12, 11, 10)-vehicle instances, respectively. Regarding SDM, our method improves the results by 7.81% and 7.79% for the same sets of instances. The 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)-vehicle instances and to less than 1% on the second set ((12, 11, 10)-vehicle instances). For the same set of instances, ALNS still has a deviation percentage of around 2%.

The good performance of our method goes along with an increase in computational time, with an average of 59.21 s for (9, 8, 7) vehicles and 83.76 s for (12, 11, 10) vehicles reported in the Hyb-Heur₁₀ column, i.e., an overall total of 71.49 s compared to 9.10 s for 100-node instances. This is mainly caused by the combinatorial explosion and the algorithm parameters since they are based on the parameters of the problem. We notice also a substantial increase in computational time of ALNS, by reaching an av-

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

Class	ALNS			SDM		Hyb-Heur ₁₀			hyb-heur ₁		GAP (%)	
	$\bar{T}(s)$	$P_{best}(\%)$	$Dev(\%)$	$T(s)$	$P(\%)$	$\bar{T}(s)$	$P_{best}(\%)$	$Dev(\%)$	$T(s)$	$P(\%)$	ALNS	SDM
<i>C100</i>	619.33	66.57	2.19	86.40	73.51	88.21	79.01	0.92	79.79	78.47	18.68	6.74
<i>C200</i>	566.36	61.46	1.81	79.50	68.78	74.93	74.77	1.04	74.19	74.14	21.67	7.79
<i>R100</i>	585.49	70.29	1.78	59.92	77.27	81.58	83.75	0.88	82.81	83.06	19.15	7.49
<i>R200</i>	589.75	70.17	2.04	77.79	76.23	80.85	83.60	0.99	85.16	82.90	19.13	8.76
<i>RC100</i>	607.04	72.46	1.72	258.62	80.30	92.83	87.05	0.94	93.90	86.48	20.13	7.70
<i>RC200</i>	633.17	73.58	1.95	175.35	80.24	84.14	87.50	0.99	89.25	86.84	18.93	8.22
<i>Mean</i>	600.19	69.09	1.92	122.93	76.05	83.76	82.61	0.96	84.18	81.98	19.62	7.79

erage of 557.68 s and 600.19 s for SET1 and SET2, respectively, i.e., a factor of more than 4 compared to 100-node instances, and almost 8.5 times more than the 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 the Hybrid Heuristic, reaching an average of 206.22 s on 200-node instances. Interestingly, computational times of SDM remarkably decrease from 290.51 s for (9, 7, 6)-vehicle instances to 122.91 s for (12, 11, 10)-vehicle instances.

5. Conclusion and perspectives

In this paper, we focused on a new variant of the Team Orienteering Problem, referred to the Synchronized Team Orienteering Problem with Time Windows. This problem was originally proposed in order to model and solve asset protection problems during out-of-control wildfires. To solve this problem, we proposed the Hybrid Heuristic method that 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 temporarily 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. Detailed computational tests prove, on the one hand, the relevance of different components of the Hybrid Heuristic, and on the other, the efficiency of our approach when compared to the literature. This study paves the way for several research directions. The first one 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 prevent them from being overwhelmed. Another promising research direction is the development 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 having funded this work. This study was carried out within 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 French National Research Agency (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] Claudia Archetti, Dominique Feillet, Alain Hertz, and Maria Grazia Speranza. The capacitated team orienteering and profitable tour problems. *Journal of the Operational Research Society*, 60(6):831–842, 2009.
- [4] Claudia Archetti, Alain Hertz, and Maria Grazia Speranza. Metaheuristics for the team orienteering problem. *Journal of Heuristics*, 13(1):49–76, 2007.
- [5] 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.
- [6] Ilaria Baffo, Pasquale Carotenuto, and Stefania Rondine. An orienteering-based approach to manage emergency situation. *Transportation research procedia*, 22:297–304, 2017.
- [7] 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.
- [8] Burcu Balcik and İhsan Yanikoğlu. A robust optimization approach for humanitarian needs assessment planning under travel time uncertainty. *European Journal of Operational Research*, 282(1):40–57, 2020.
- [9] 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.
- [10] Nicola Bianchessi, Renata Mansini, and M Grazia Speranza. A branch-and-cut algorithm for the team orienteering problem. *International Transactions in Operational Research*, 25(2):627–635, 2018.
- [11] Sylvain Boussier, Dominique Feillet, and Michel Gendreau. An exact algorithm for team orienteering problems. *4or*, 5(3):211–230, 2007.
- [12] 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.
- [13] I-Ming Chao, Bruce L Golden, and Edward A Waisl. The team orienteering problem. *European Journal of Operational Research*, 88(3):464–474, 1996.
- [14] Teodor Gabriel Crainic, Nicoletta Ricciardi, and Giovanni Storchi. Models for evaluating and planning city logistics systems. *Transportation science*, 43(4):432–454, 2009.
- [15] Duc-Cuong Dang, Rym Nesrine Guibadj, and Aziz Moukrim. An effective pso-inspired algorithm for the team orienteering problem. *European Journal of Operational Research*, 229(2):332–344, 2013.
- [16] Michael Drexler. Synchronization in vehicle routing—a survey of vrps with multiple synchronization constraints. *Transportation Science*, 46(3):297–316, 2012.
- [17] Racha El-Hajj, Duc-Cuong Dang, and Aziz Moukrim. Solving the team orienteering problem with cutting planes. *Computers & Operations Research*, 74:21–30, 2016.
- [18] Thomas A Feo and Mauricio GC Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133, 1995.
- [19] 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.

- [20] Aldy Gunawan, Hoong Chuin Lau, and Pieter Vansteenwegen. Orienteering problem: A survey of recent variants, solution approaches and applications. European Journal of Operational Research, 255(2):315–332, 2016.
- [21] Syrine Roufaïda 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.
- [22] Qian Hu and Andrew Lim. An iterative three-component heuristic for the team orienteering problem with time windows. European Journal of Operational Research, 232(2):276–286, 2014.
- [23] 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.
- [24] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search. In Handbook of metaheuristics, pages 320–353. Springer, 2003.
- [25] 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.
- [26] Dian Nuraiman, Melih Ozlen, and John Hearne. A spatial decomposition based math-heuristic approach to the asset protection problem. Operations Research Perspectives, 7, 2020.
- [27] 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.
- [28] Iman Roozbeh, John W Hearne, and Delaram Pahlevani. A solution approach to the orienteering problem with time windows and synchronisation constraints. Heliyon, 6(6):e04202, 2020.
- [29] 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.
- [30] 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.
- [31] 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.
- [32] Pieter Vansteenwegen, Wouter Souffriau, Greet Vanden Berghe, and Dirk Van Oudheusden. Iterated local search for the team orienteering problem with time windows. Computers & Operations Research, 36(12):3281–3290, 2009.
- [33] 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.
- [34] Ala-Eddine Yahiaoui, Aziz Moukrim, and Mehdi Serairi. The clustered team orienteering problem. Computers & Operations Research, 111:386–399, 2019.