Scheduling Plans of Tasks
Internship report. Supervisors: Amal El Fallah Seghrouchni, Safia-Kedad Sidhoum
Davide Andrea Guastella

Abstract—We present a heuristic algorithm for solving the problem of scheduling plans of tasks. The plans are ordered vectors of tasks, and tasks are basic operations carried out by resources. Plans are tied by temporal, precedence and resource constraints that makes the scheduling problem hard to solve in polynomial time.

The proposed heuristic, that has a polynomial worst-case time complexity, searches for a feasible schedule that maximize the number of plans scheduled, along a fixed time window, with respect to temporal, precedence and resource constraints.

I. INTRODUCTION
Scheduling is a decision-making process that is used on a regular basis in many manufacturing and services industries [1]. It deals with the allocation of resources to tasks over given time periods and its goal is to optimize one or more objectives.

From a theoretical point of view, a scheduling problem is a constrained combinatorial optimization problem where a set of task must be ordered in such a way that all these are arranged, according to one or more constraints, to constitute a schedule that minimize or maximize a given objective function.

One of the most popular scheduling problem is the resource-constrained project scheduling problem (RCPSP) [2]: the objective of resource-constrained scheduling consists in developing a schedule such that a set of tasks is completed as early as possible considering both the precedence relationships and the restricted availabilities of resources.

The scheduling problem we are facing can be thought as a particular case of the RCPSP problem: in our case we are facing the problem of scheduling plans of tasks subject to both precedence, resource and temporal constraints. Also, rather than scheduling tasks as in the RCPSP, our problem aims at selecting a maximum number of plans.

II. CASE STUDIES
Nowadays, airborne platforms such as Remote Piloted Air Vehicles (RPAS) are employed in different scenario including conflicts, surveillance and rescue [3]. In these scenario, airborne platforms operate in highly dynamic environments with a low predictability. In this context, onboard instruments (i.e. sensors) allow the platform, hence the mission manager, to collect knowledge from the field. Sensors carried by RPAS are now able to perform a large panel of functions such as image acquisition, spectrum analysis, and object tracking [4]. All these sensors play a major role in operation and their optimization has become essential.

Because of the criticality of the context and the mission’s objectives, it is important to develop a method that orchestrates the operations conducted by the sensors, such that the mission is accomplished correctly and by satisfying all the constraints. Moreover, due to both critical contexts and dynamic environment, it is important to orchestrate the operations of the sensors within a relatively short time.

As further case study, consider the in-flight airplane safety [5]. In this context there is a need for detecting and resolving data errors which could come from faulty sensor measurements, inaccurate data processing, or poor information transmission, that can lead to catastrophic accidents as in the case of the Air France 447’s accident. Our scheduling technique can be employed as a fault recovery technique: for example, once the fault has been detected by the on-board sensors, a system scheduler could discards the remaining flight plans, and schedules a set of emergency plans into the current flight schedule. These emergency plans are strictly time constrained, and their orchestration necessarily needs to lead to a schedule which can guarantee the safety of passengers.

III. DEFINITIONS

Definition 1 (Plan of Tasks). A plan of tasks \( \Pi_k \) is a partial ordering of tasks \( J^k_i \) to address a specific goal. Each plan \( \Pi_k \) is characterized by a priority value \( \alpha_k \). The structure of the plan is depicted by an activity-on-node (AON) network where the nodes and the arcs represent the tasks and the precedence relations respectively [6]. The precedence relations are described by the notation \( \prec \). For example, given two plans \( \Pi_k \) and \( \Pi_j \), \( \Pi_k \prec \Pi_j \) indicates that the plan \( \Pi_k \) must be scheduled before \( \Pi_j \).

A plan is defined by the following notation:

\[ \Pi_k = (J^k_1, \ldots, J^k_{n_k}) \]

where \( k \) is an arbitrary index for the plan, \( n_k \) is the number of tasks in the plan \( \Pi_k \) and \( J^k_i \), with \( 0 \leq i \leq n_k \) is the \( i \)-th task of the plan \( \Pi_k \). The set of tasks is topologically sorted (see 3).

A plan \( \Pi_k \) could not be scheduled if at least one task \( J^k_i \in \Pi_k \) could not be scheduled due to unsatisfied constraints. A plan \( \Pi_k \) is characterized by a set of resources \( \mathcal{R}^k \subseteq \mathcal{R} \).

Definition 2. A graph \( G = (V, E) \) consists of a set of vertices \( V \), and a set of edges \( E \subseteq V \times V \). In a directed graph the edges are directed from one vertex to another. A directed acyclic graph (or DAG) is a directed graph with no directed cycles: a directed cycle is a path that starts from any vertex and ends in it.

Definition 3 (Topological Sort). A topological sort of a directed acyclic graph (DAG) \( G = (V, E) \) is a linear ordering of all its vertices such that if the graph \( G \) contains an edge
\((u, v)\) then \(u\) appears before \(v\) in the ordering \([7, 8]\). If the graph contains a cycle, then no linear ordering is possible.

For example, in Figure 1 a directed acyclic graph is showed, for which a topological sorting can be found, since it has no cycles. Figure 2 shows a possible topological sorting for the graph in Figure 1, where the different colors depict the different frontiers in the corresponding DFS graph.

**Definition 4** (Frontier). A frontier \(f\) for a graph \(G = (V, E)\) is a set of nodes \(f \subseteq V\) such that the maximum distance between each node \(u \in f\) and the root node in the corresponding DFS graph is the same.

![Fig. 1: A Directed Acyclic Graph.](image1)

![Fig. 2: A possible topological sorting for the DAG in Figure 1.](image2)

**Definition 5** (Task). A task \(J^k_i\) is an atomic operation that a subset of resources must execute. Each task \(J^k_i\) belongs uniquely to one plan \(\Pi_k\). Tasks are characterized by the following elements:
- a processing time \(p^k_i \in \mathbb{N}_{0}\), \(p^k_i \geq 1\);
- a release date \(r^k_i \in \mathbb{N}\);
- a due date \(d^k_i \in \mathbb{N}_{0}\);
- a time lag \(\delta^k_{ij} \in \mathbb{N}\), if \(J^k_i \prec J^k_j\);
- a set of resources \(R^k_i = \{\rho_1, \ldots, \rho_n\}\), \(R^k_i \subseteq \mathcal{R}\), which the task \(J^k_i\) is assigned to.

The tasks within a plan can be dependent by a precedence graph that provides the precedence constraints between the tasks. As for the plans, the precedence relations between tasks are described by the notation \(\prec\). For example, given two tasks \(J^k_i\) and \(J^k_j\) belonging to the same plan \(\Pi_k\), \(J^k_i \prec J^k_j\) indicates that the task \(J^k_i\) precedes the execution of the task \(J^k_j\).

**Definition 6** (Time lag). A time lag \(\delta^k_{ij}\) is a fixed amount of time which separates the finishing and starting times of a pair of tasks \(J^k_i\) and \(J^k_j\) respectively. By default, it is 0.

In this paper, we assume that there is no preemption in the task execution. Therefore, the execution of the tasks can not be interrupted once started. Moreover, we assume that each task could consume more than one resource during its execution.

**Definition 7** (Resource). A resource \(\rho\) is any hardware or software tool that tasks can use to handle information. A resource \(\rho\) has a limited availability value \(B_p\). Resources are typically distinct in renewable and nonrenewable [6]: renewable resources have a fixed value of availability in each time period, while nonrenewable resources have a fixed value of availability along the entire project’s time horizon. In the addressed scheduling problem, we deal with renewable resources with a fixed availability value of \(B_p = 1\), \(\forall p \in \mathcal{R}\), in each time period. In particular, resources with a fixed availability value of \(B_p = 1\) are also called unary or disjunctive resources.

**Definition 8** (Resource availability). The availability \(B_{\rho} \in \mathbb{N}_{0}\) of a resource \(\rho\) represents the maximum value of availability of the resource \(\rho\) in each time period.

The abstract amount of usage of the resource \(\rho\) by the task \(J^k_i\), in each time period, is represented by \(b^k_{ij} = [0, 1]\).

In this work we are concerned with solving the problem of scheduling plans of tasks by using a heuristic that maximizes the number of plans scheduled in a fixed time window, taking into account precedence, time, and resource constraints.

### IV. Problem statement

#### A. Constraints

1) **Temporal constraints**: When a plan \(\Pi_k\) is scheduled, for each task \(J^k_i \in \Pi_k\) both the starting time \(s^k_i\) and the completion time \(C^k_i = s^k_i + p^k_i\) must be inside the temporal window \([W_s, W_e]\) and also inside the temporal window \([r^k_i, d^k_i]\), where

\[
(r^k_i \geq W_s) \land (d^k_i \leq W_e), \ \forall J^k_i \in \Pi_k.
\]

The value of \(s^k_i\) for a task \(J^k_i \in \Pi_k\) is calculated as the maximum value between \(W_s\), the release date \(r^k_i\) of the task \(J^k_i\) augmented by the time lag \(\delta^k_{ij}\), and the maximum completion time \(C^k_j\) for each predecessor \(J^k_j \in \Pi_k\). Formally,

\[
s^k_i = \max(W_s, r^k_i, \max_{j \in \text{pred}^k_i}(C^k_j + \delta^k_{ij})), \tag{1}
\]

where \(\text{pred}^k_i\) is the set of predecessors of the task \(J^k_i\).

Given a set \(\mathcal{P}\) of plans, the plan \(\Pi_k \in \mathcal{P}\) satisfies the temporal constraints if

\[
\exists s^k_i \forall J^k_i \in \Pi_k : (s^k_i \geq r^k_i) \land (s^k_i + p^k_i \leq d^k_i).
\]

2) **Precedence constraints**: In order to maintain the precedence constraints between plans, a set \(\mathcal{P}\) of plans to be scheduled is topologically ordered, since the precedence between plans can be represented as an acyclic directed graph (DAG), as showed in Figure 1. In our context, a graph \(G\) is a tuple \((V, E)\) where \(V\) is a set of vertices and \(v_k \in V\) represents a plan \(\Pi_k \in \mathcal{P}\), and an arc \((u, v)\) represents a precedence constraint between the plans that are represented by nodes \(u\) and \(v\) respectively.
3) Resource constraints: The scheduling problem we address assumes that each task $J_i^k \in \Pi_k$ which has a feasible starting time $s_i^k$, can be executed by a subset $R_k^i \subseteq R_k$ of resources if in the time period $[s_i^k, C_i^k]$ the utilization of each resource $\rho \in R_k^i$ does not exceed its availability $B_{\rho}$.

B. Classification and mathematical formulation

A typical formal way to describe the scheduling problems is by using the three-field classification $[\alpha, \beta, \gamma]$ introduced by Graham et al. [9], where

- $\alpha$ specifies the machine environment,
- $\beta$ specifies the characteristics of the activities,
- $\gamma$ describes the objective function(s).

An extension of the Graham’s classification has been proposed in order to provide a more accurate way to formally describe the scheduling problems [10]. According to the extended notation, the plans of tasks scheduling problem can be stated as

$$PS1, 1, 1 \mid prec_P; prec_J; [W_s, W_e]; [r_i^k, d_i^k]; \delta_{ij}^{k}, \max_k \sum \alpha_k x_k$$

where $PSm, 1, 1 \in \alpha$ indicates a resource environment for a project scheduling problem with $m$ resources, a maximum availability of 1 unit per time for each resource, and a resource utilization, by each task, of at most 1 unit per time period. $prec_P; prec_J; [W_s, W_e]; [r_i^k, d_i^k]; \delta_{ij}^{k} \in \beta$ depicts the characteristics of both plans and tasks; $prec_P$ indicates a precedence constraints between plans; $prec_J$ indicates a precedence constraints between tasks; $[W_s, W_e]$ and $[r_i^k, d_i^k]$ indicates, for each task $J_i^k \in \Pi_k$, the time constraints discussed in IV-A1; $\delta_{ij}^{k}$ indicates that each task $J_i^k$ can have a time lag between its starting time $s_i^k$ and the completion time $C_i^k$ of its predecessor.

The mathematical formulations for the RCPSP can be conveniently employed to model the objective function with respect to the temporal, precedence and resource constraints [11]. The proposed formulation is based on time discretization for describing the usage of the resources and the processing of the tasks over time. In our context the time horizon, divided into unitary time periods, is a fixed time window $[W_s, W_e]$, where the maximum number of plans has to be scheduled. Given a plan $\Pi_k \in \mathcal{P}$ and a task $J_i^k \in \Pi_k$, and a time instant $t \in [r_i^k, d_i^k]$, let $y_{it}^k$ be a boolean variable that indicates whether the task $J_i^k$ starts exactly at time $t$. Also, let $x_k$ be a boolean variable that indicates whether a plan $\Pi_k \in \mathcal{P}$ is executed. Thus, a plan $\Pi_k \in \mathcal{P}$ is executed if

$$\sum_{i=1}^{n_k} \sum_{t=r_i^k}^{d_i^k} y_{it}^k = n_k \Rightarrow x_k = 1,$$

where $n_k$ is the number of tasks in $\Pi_k$. If a task $J_i^k$ has a feasible starting time $s_i^k$ equals to $t$, then $y_{it}^k = 1$ for a time instant $t \in [r_i^k, d_i^k]$. Thus, if the sum

$$\sum_{i=1}^{n_k} \sum_{t=r_i^k}^{d_i^k} y_{it}^k$$

is equal to $n_k$, then $\Pi_k$ is executed. Therefore, the variable $x_k$ is equal to 1. By using this notation, the objective function to maximize can be formulated as:

$$\max_k \sum_{k=1}^{K} \alpha_k x_k$$

such that,

$$\begin{align*}
& n_k \sum_{i=1}^{n_k} \sum_{t=r_i^k}^{d_i^k} y_{it}^k = n_k \Rightarrow x_k = 1 \quad \forall \Pi_k \in \mathcal{P} \\
& (t + \delta_{ij}^{k} \geq r_j^k) \land (t + \delta_{ij}^{k} + p_{i}^{k} \leq d_{i}^{k}) \Rightarrow y_{it}^{k} = 1 \quad \forall t \in [W_s, W_e], \\
& \sum_{t} (y_{it}^{k} - y_{jt}^{k}) \geq p_{i}^{k} + \delta_{ij}^{k} \quad \text{if } J_i^k \prec J_j^k, \\
& \forall J_i^k, J_j^k \in \Pi_k, i \neq j \\
& \sum_{i=1}^{n_k} \sum_{t=r_i^k}^{d_i^k} y_{it}^k \leq B_{\rho} \quad \forall \rho \in \mathcal{R}, \forall t \in [W_s, W_e]
\end{align*}$$

Constraint (2) imposes that for each plan $\Pi_k \in \mathcal{P}$ all the tasks $J_i^k \in \Pi_k$ have a starting time assigned. Constraint (3) imposes that each task $J_i^k$ has a starting time and a completion time within the time window $[r_i^k, d_i^k]$. Constraints (4) and (5) impose respectively the precedence and resource constraints.

V. PROPOSED METHOD

In this section is discussed a greedy heuristic for the problem of scheduling plans of tasks.

Definition 9 (Schedule). A schedule $S$ is a vector $(s_i^k)$, $i = 1, \ldots, n_k$, $k = 1, \ldots, K$ such that $s_i^k \in S$ is a starting time for the task $J_i^k$. A schedule represents a solution for the scheduling problem if it satisfies the scheduling constraints.

Given a temporal window $W = [W_s, W_e]$, the goal of the proposed heuristic is to create a feasible schedule that contains the maximum number of plans of tasks, inside the temporal window $[W_s, W_e]$, regarding their weight, such that each scheduled plan $\Pi_k$ satisfies the precedence, temporal and resource constraints discussed in IV. The set $\mathcal{P}$ of plans given as input to the algorithm is preventively sorted in order to maintain their precedence relations, and also to evaluate, at each iteration of the algorithm, the plan with the highest priority value. The feasibility of the plans is evaluated in a schedule $S_{w}$ until no plans remains to schedule, or there is no plan that can be scheduled due to a violation of the constraints.

Each plan $\Pi_k \in \mathcal{P}$ could have precedence relations between its tasks. The topological sorting of the tasks ensures that the execution of the tasks is done with respect to their precedence relations. Example 1 shows an example of feasible schedule where a precedence relation occurs between the tasks of a plan.
Figure 3 shows a Gantt diagram for a feasible schedule for a set of plans \( \mathcal{P} = \{\Pi_1, \Pi_2\} \), which parameters are listed in Table I. A precedence constraint occurs between the tasks \( J^1_2 \) and \( J^2_2 \), both belonging to the plan \( \Pi_2 \).

| \( J^1_2 \) | \( r^1_2 \) | \( d^1_2 \) | \( p^1_2 \) | \( \delta^1_2 \) | \( R^1_2 \) | \( J^2_1 \) | \( J^2_2 \) |
|---|---|---|---|---|---|---|---|
| 2 | 7 | 3 | 0 | \( J^1_2 < J^2_2 \) |

TABLE I: The parameters of the tasks of the plans \( \Pi_1 \) and \( \Pi_2 \) in the Example 1.

Example 1. For a given DAG, more than one topological sort could lead to different schedules. For this reason, the plans belonging to the same frontier are sorted by their priority value, since for these plans the order in which they are scheduled is irrelevant according to the precedence constraints. Moreover, different plans may have the same priority value: in this case, the plans must be sorted according to a different criterion described in the following.

Given a feasible schedule \( S_w \) and a subset \( \mathcal{P}_{\alpha_k} \subseteq \mathcal{P} \) of plans having the same priority value \( \alpha_k \), in order to maximize the resource utilization for each task \( J^k_i \in \Pi_k \in \mathcal{P}_{\alpha_k} \), we calculate an idle time between each task \( J^k_i \in \Pi_k \) and its predecessor, in the same resource.

Definition 10 (Idle time). Let \( S \) be a feasible schedule containing two plans \( \Pi_1, \Pi_p \), \( p \neq k \). Let \( J^k_i \in \Pi_k \) and \( J^p_j \in \Pi_p \) be two tasks scheduled on the same resource \( \rho \in \mathcal{R} \), where \( s^k_i \geq C^p_j \). If there is at least one plan \( \Pi_q \) schedulable in \( S \) such that \( J^q_q \in \Pi_q \), executed by the same resource \( \rho \in \mathcal{R} \) such that

\[
(C^p_j \leq s^q_q < s^k_i) \lor (C^p_j < C^q_q \leq s^k_i),
\]

the time window \([C^p_j, s^k_i]\) is said to be an idle time.

In Figure 4 is showed an idle time between two tasks.

![Fig. 3: A feasible schedule for the set \( \mathcal{P} \) of two plans. A precedence constraint occurs between \( J^1_2 \) and \( J^2_2 \).](image)

![Fig. 4: The time window \([C^p_j, s^k_i]\) is an idle time because \( J^q_q \in \Pi_q \) can be scheduled in this interval.](image)

TABLE II: The parameters of the tasks of a set \( \mathcal{P} \) of 4 plans.

Let \( S \) be a feasible schedule, and \( \mathcal{P}_{\alpha_k} \) a set of plans having each one priority \( \alpha_k \). The plans of \( \mathcal{P}_{\alpha_k} \) are scheduled in order to minimize the presence of idle times. This is done by choosing iteratively the plans who minimize the following quantity:

\[
\sum_{i} s^k_i - C^{\Gamma^k_i}_{(\rho_i)}, \forall J^k_i \in \Pi_k \in \mathcal{P}_{\alpha_k}
\]

where \( C^{\Gamma^k_i}_{(\rho_i)} \) is the completion time of the task \( \Gamma^k_i \) in the same resource \( \rho_i \) executing the task \( J^k_i \).

Example 2. Let \( \mathcal{P} = \{\Pi_1, \Pi_2, \Pi_3, \Pi_4\} \) be a set of plans to schedule, which information are listed in Table II, and let \( S_w \) be a feasible schedule where \( \Pi_1 \) and \( \Pi_2 \) have been scheduled (Figure 5a). Let \( \mathcal{P}_l = \{\Pi_3, \Pi_4\} \) be a set of two plans such that \( \Pi_3 \) and \( \Pi_4 \) have priority values \( \alpha_3 = l \) and \( \alpha_4 = l \) respectively. As showed in Figure 5b-5c, both plans \( \Pi_3 \) and \( \Pi_4 \) generate two feasible schedules.

The difference between the starting time \( s^1_3 \) and its predecessor (in this case \( W_1 \) since there is no predecessor of \( J^1_3 \)) is equal to 2. Within \( \mathcal{P}_l \), the plan \( \Pi_4 \) has a task \( J^4_1 \) that can be scheduled at \( s^1_3 = 3 < s^2_1 \), where all the constraints are satisfied, and both \( J^1_3 \) and \( J^1_3 \) demand the same resource. Moreover, scheduling the plan \( \Pi_4 \) before \( \Pi_3 \) leads to a schedule where no idle times are generated. Therefore, plan \( \Pi_4 \) is scheduled before plan \( \Pi_3 \).

The scheduling algorithm operates in a similar way as the serial scheduling scheme proposed in [11]. In our context, we assume that each resource has a maximum availability of 1 in each time period. In this case, each resource can execute one task at a time. The idea of the proposed heuristic is to schedule all the plans as soon as possible by respecting the resource, precedence and temporal constraints. The insertion of the plans into a schedule is done by handling an event list \( \mathcal{E} \). An event is a 4-tuple \((i(e), S(e), C(e), (b_p(e))_{v \in \mathcal{R}})\):

- \( i(e) \) denotes the time instant of the event;
- \( S(e) \) is the set of tasks starting exactly at time \( t(e) \);
- \( C(e) \) is the set of tasks finishing exactly at time \( t(e) \);
- \( (b_p(e))_{v \in \mathcal{R}} \) is the set of tasks that can be scheduled in the event.
The proposed heuristic (Algorithm 1) \((Build Schedule)\) takes input a set \(P\) of plans and a set \(R\) of resources, and it gives as output a feasible schedule \(S_t\), that is, a vector of starting times \(s^i_t\) for each scheduled task \(J^i_k \in \Pi_k\) such that the constraints discussed in the previous section are met.

At the beginning, the algorithm initializes the used data structures. At line 1 and 2 respectively, two set \(P_s\) and \(P_f\) are initialized: they will be used to store respectively the successfully scheduled plans and the discarded (unscheduled) plans. Then, two empty schedules \(S_l\) and \(S_w\) are initialized. They will contain respectively the last feasible schedule found and the current working schedule. The working schedule is a schedule used for testing the feasibility of the plans during the search. Finally, an event list \(\mathcal{E}\mathcal{L}\) is initialized at line 5. The first step of the algorithm is to sort the set \(P\) of plans according to both precedence relations and priority values (line 6). This is done by Algorithm 2, that returns a sorted list \(P_{\alpha}\) of plans.

At line 9 the algorithm get the number of plans in \(P\), \(|P_{\alpha}|\). If the value of \(\alpha_k\) is unique among the values of priority of the plans in \(P_{\alpha}\), the algorithm tries to schedule the plan (line 11), otherwise a set \(P_{\alpha_k} \subset P\) of plans is calculated, containing all the plans that have priority value \(\alpha_k\). All the plans in \(P_{\alpha_k}\) are then scheduled by Algorithm 5. Thus, the plans successfully scheduled are added into \(P_s\) (line 20), and the plans not scheduled are removed from the working schedule (line 22). Finally, the last feasible schedule \(S_l\) is updated (line 24).

Algorithm 1 takes at most \(O(K^3 n^3)\) time, where \(K = |P|\) is the number of plans to schedule.

All the data structures used for storing plans, \(P\), \(P_u\), \(P_s\) and \(P_f\), are implemented as self-balancing binary search trees to ensure insertion, removal and get operations in logarithmic time [7]. At line 5 the event list \(\mathcal{E}\mathcal{L}\) is initialized. As the data structures used for storing plans, the event list is implemented as self-balancing binary search tree. At line 6 the plans are sorted by the Algorithm 2.

The \(Get\) operations at line 8 takes logarithmic time with respect to the number of elements in \(P_u\).

At line 9 the algorithm get the number of plans in \(P_u\) that have priority value \(\alpha_k\). We use an auxiliary data structure to store this information and get the value in \(O(1)\).

The \(Add\) and \(Remove\) operations at line 13 and 15 respectively require both \(O(\log K)\) time. Getting the plans with priority \(\alpha_k\) (line 18) requires constant time, since a hash map is used to store the plans with the same priority value: the key of a record is a value of priority \(\alpha_s\), and the value is a list of plans having each one \(\alpha_k\) as priority value.

**A. Description of the heuristic**

- \(C(e)\) is the set of tasks completing exactly at time \(t(e)\):
- \(b_{\rho}(e)\) represents the usage of resource \(\rho\) during interval \([t(e), t(\text{next}_e)]\), where \(\text{next}_e\) is the first event following \(e\) in \(\mathcal{E}\mathcal{L}\).

When a feasible task \(J^i_k\) is scheduled and inserted into \(S(e)\) for an event \(e \in \mathcal{E}\mathcal{L}\), the usage of the resource \(b_{\rho}(e)\) is set to 1. The resource constraints for the scheduling problem we address assumes that no task can be scheduled at \(t(e)\) if \(b_{\rho}(e) = 1\).
The Add operation at line 20 requires $O(K \log K)$ time. The Remove operations at line 21 and 22 require both $O(K \log K)$ time.

Algorithm 1 BuildSchedule

Require: A set of plans $\mathcal{P}$
Ensure: A feasible schedule $S_t$, a set $\mathcal{P}_s$ of scheduled plans, a set $\mathcal{P}_f$ of unscheduled plans
1: $\mathcal{P}_s \leftarrow \{\text{Scheduled plans}\}$
2: $\mathcal{P}_f \leftarrow \{\text{Discarded plans}\}$
3: $S_t \leftarrow \{\text{Feasible schedule}\}$
4: $\mathcal{P}_w \leftarrow \{\text{Working schedule}\}$
5: $\mathcal{E} \mathcal{L} \leftarrow \{\}$
6: $\mathcal{P}_u \leftarrow \text{SortPlans}(\mathcal{P})$
7: while $\mathcal{P}_u \neq \emptyset$ do
8:   $\Pi_k \leftarrow \text{Get}(\mathcal{P}_u)$
9:   $N_{\alpha_k} \leftarrow$ number of plans in $\mathcal{P}_u$ with priority $\alpha_k$
10:  if $N_{\alpha_k} = 1$ then
11:    $\text{success} \leftarrow \text{SchedulePlan}(\Pi_k, S_w, \mathcal{E} \mathcal{L})$
12:    if success then
13:       $\text{Add}(\Pi_k, \mathcal{P}_s)$
14:    else
15:       $\text{Remove}(\Pi_k, S_w)$
16:  end if
17:  else
18:   $\mathcal{P}_{\alpha_k} \leftarrow$ plans of $\mathcal{P}$ with priority $\alpha_k$
19:   $U \leftarrow \text{SchedulePlanSet}(\mathcal{P}_{\alpha_k}, S_w, \mathcal{E} \mathcal{L})$
20:   $\text{Add}(\mathcal{P}_{\alpha_k} \setminus U, \mathcal{P}_s)$
21:   $\text{Remove}(\mathcal{P}_{\alpha_k}, \mathcal{P}_u)$
22:   $\text{Remove}(U, S_w)$
23:  end if
24: $S_t \leftarrow S_w$
25: $\mathcal{P}_f = \mathcal{P} \setminus \mathcal{P}_s$
26: end while

Algorithm 2 SortPlans

Require: A set of plans $\mathcal{P}$
Ensure: A sorted set of plans $\mathcal{P}_u$
1: $\mathcal{P}_t \leftarrow \text{TopologicalSort}(\mathcal{P})$
2: compute $F_i$ for $i = 1, \ldots, f$
3: $\mathcal{P}_u = \emptyset$
4: for $i = 1, \ldots, f$ do
5:   sort $F_i$ according to $\alpha$
6:   $\text{Add}(F_i, \mathcal{P}_u)$
7: end for
8: return $\mathcal{P}_u$

Algorithm 3 SchedulePlan

Require: A plan $\Pi_k$, a schedule $S_w$, an event list $\mathcal{E} \mathcal{L}$
Ensure: true if $\Pi_k$ has been scheduled in $S_w$, false otherwise.
1: for all task $J_k^i \in \Pi_k$ do
2:   $\text{success} \leftarrow \text{scheduleTask}(J_k^i, S_w, \mathcal{E} \mathcal{L})$
3: if not success then
4:   mark $\Pi_k$ as unschedulable
5: return false
6: end if
7: end for
8: return true
plan $\Pi_k$ are removed from each event $e$ in the event list $\mathcal{EL}$ (line 14), the plan $\Pi_k$ is marked as unschedulable (line 15) and false is returned.

Once the constraints are verified for $e$ at line 13, the algorithm proceeds with the insertion of the task $J^k_t$ into the working schedule (line 18), and into the event list (line 19). Then, at line 21, the event $e$ is added into the event list $\mathcal{EL}$ if it is not already in. The steps (23-31) update the event list. The last step consists in updating the resources usage by the scheduled tasks (line 32).

Algorithm 4 has a time complexity of $O(n^2)$, where $n = \sum_k n_k$. Searching for a feasible starting time $s^k_i$ (line 1) requires at most $O(n-1)$ time if task $J^k_t$ has $n-1$ predecessors. Searching for the event $e$ such that $t(e) = s^k_i$ (line 2) requires at most $O(\log |\mathcal{EL}|)$ since a balanced tree data structure is used for the event list.

In the worst case, for a set $\mathcal{P}$ of plans with cardinality $|\mathcal{P}| = K$, the maximum number of events in $\mathcal{EL}$ is $2n$ if we suppose that all the tasks are scheduled consecutively with a minimal lag between each task, that is

$$|S(e)| = 1 \land |C(e)| = 0 \lor$$
$$|S(e)| = 0 \land |C(e)| = 1, \forall e \in \mathcal{EL}.$$  

The while loop (lines 4-12) iterates over the event list $\mathcal{EL}$. Since there are less than $2n$ events, the while loop takes $O(n)$ time.

Searching for the event following $f$ in $\mathcal{EL}$ (line 5) takes at most $O(\log |\mathcal{EL}|)$ time. Checking if the temporal constraints are satisfied at $t(e)$ (line 6) takes $O(1)$ time. The remaining operations inside the loop require constant time.

Removing all the tasks $J^k_t \in \Pi_k$ from each event $e \in \mathcal{EL}$ (line 14) takes $O(n_k \log |\mathcal{EL}|)$.

Adding the task $J^k_t$ to the schedule $S_w$ (line 18) requires at most $O(\log n)$ since an efficient balanced tree data structure is used for the schedules. Adding the task $J^k_t$ to the set $S_c$ (line 19) requires constant time.

Adding the event $e$ to the event list $\mathcal{EL}$ (line 21) requires $O(\log |\mathcal{EL}|)$ time.

The event list update takes $O(\log |\mathcal{EL}|)$ if the event $g$ has to be inserted in $\mathcal{EL}$ (lines 26,29), otherwise the insertion of $J^k_t$ into $C(f)$ (line 24) requires $O(1)$ time.

Algorithm 5 (SchedulePlanSet) schedules a set $\mathcal{P}_{\alpha_k}$ of plans such that each plan $\Pi_k \in \mathcal{P}_{\alpha_k}$ has a priority value $\alpha_k$. The idea of the algorithm is to find, at each iteration, the plan that minimizes the sum of the size of the idle time windows generated by its tasks. The algorithm starts by creating a copy $V$ of the plan set $\mathcal{P}_{\alpha_k}$ (line 1). The while loop (lines 3-25) iterates until there is no plan left in $V$ to schedule. At each iteration, two variables minIdleTime and bestPlan are used to keep the next candidate plan to schedule. The loop at (6-19) iterates over the plans in $V$ and searches for the candidate plan to schedule. Each plan $\Pi_k$ is scheduled into a temporary schedule $S_{temp}$ (line 9). At each iteration, the algorithm chooses the plan $\Pi_k \in \mathcal{P}_{\alpha_k}$ that minimizes the sum of the time differences between $t(e)$ and $t(pred_e)$ (line 11), for each event $e$ such that $J^k_t \in S(e), \forall J^k_t \in \Pi_k$.

Algorithm 4 ScheduleTask

Require: A task $J^k_t$, a schedule $S_w$, an event list $\mathcal{EL}$
Ensure: true if $J^k_t$ has been scheduled in $S_w$, false otherwise.

1: $s^k_i = \max(W_i, r^k_t, \max_j \in pred_i(C^k_j + \delta^k_j))$
2: $e \leftarrow \text{GetEvent}(s^k_i, \mathcal{EL})$
3: $\mu \leftarrow p^k_t; f \leftarrow e$
4: while $(f$ is not the last event of $\mathcal{EL}) \land (\mu > 0)$
5: \hspace{1em} $g \leftarrow next_f$
6: \hspace{2em} if $(b_p(f) = 0, \forall p \in R_k) \land$
7: \hspace{2em} \hspace{1em} $(\text{CheckConstraints}(t(e), J^k_t))$
8: \hspace{2em} \hspace{1em} $\mu \leftarrow \max(0, \mu - t(g) + t(f))$
9: \hspace{2em} \hspace{1em} $f \leftarrow g$
10: \hspace{1em} else
11: \hspace{2em} $\mu \leftarrow p^k_t; e \leftarrow g; f \leftarrow g$
12: \hspace{1em} end if
13: \hspace{1em} $\mu \leftarrow \max(0, \mu - t(g) + t(f))$
14: \hspace{1em} $f \leftarrow g$
15: \hspace{1em} else
16: return false
17: end if
18: $\mu \leftarrow p^k_t; S_w \leftarrow S_e$
19: Add($J^k_t, S_w$)
20: if $e \notin \mathcal{EL}$ then
21: \hspace{1em} $\mathcal{EL} \leftarrow \mathcal{EL} \cup e$
22: end if
23: if $t(e) + p^k_t = t(f)$ then
24: \hspace{1em} Add($J^k_t, C(f)$)
25: else if $t(e) + p^k_t > t(f)$ then
26: \hspace{2em} insert $g \leftarrow (t(e) + p^k_t, \emptyset, \{t\}, \{b_p(f)\}_{\forall p \in R_k})$ in $\mathcal{EL}$
27: \hspace{2em} $f \leftarrow g$
28: else
29: \hspace{2em} insert $g \leftarrow (t(e) + p^k_t, \emptyset, \{t\}, \{b_p(pred_f)\}_{\forall p \in R_k})$ in $\mathcal{EL}$
30: \hspace{2em} $f \leftarrow g$
31: end if
32: if $b_p(g) = 1, \forall p \in R_k, \forall g$ between $e$ and $pred_f$ in $\mathcal{EL}$
33: return true
34: end if

Algorithm 5 takes at most $O(K^2 n^3)$ time. The main loop (lines 3-25) is repeated until there is some plan left to schedule, and it is repeated at most $O(K)$ time. The inner loop (lines 6-19) schedules each plan $\Pi_k \in \mathcal{P}_{\alpha_k}$ and keeps the plan which minimize the idle times between each task and its predecessor. The inner loop takes $O(K \alpha_k)$ time.

Algorithm 6 (CheckConstraints) checks if a starting time $t(e)$ satisfies the temporal constraints, discussed in IV-A1, for a task $J^k_t$. If the temporal constraints are satisfied, the algorithm returns true.

Algorithm 6 requires constant time.

In Table III are resumed the functions employed by the proposed method, together with a short description and their time complexity.

Example 3. In this example we show how different plans are scheduled according to the proposed heuristic. We start
Algorithm 5 SchedulePlanSet

Require: A set \( \mathcal{P}_{o_k} \subset \mathcal{P} \) of plans, a schedule \( S_w \), an event list \( \mathcal{E} \mathcal{L} \)

Ensure: A set \( \mathcal{U} \) of unscheduled plans
1: \( V \leftarrow \mathcal{P}_{o_k} \)
2: \( \mathcal{U} \leftarrow 0 \)
3: while \( V \neq 0 \) do
4: \( \text{minIdleTime} \leftarrow +\infty \)
5: \( \text{bestPlan} \leftarrow 0 \)
6: for all \( \mathcal{J}_k \in \mathcal{P}_{o_k} \) do
7: \( S_{\text{temp}} \leftarrow S_w \)
8: \( \mathcal{E}_{\text{temp}} \leftarrow \mathcal{E} \mathcal{L} \)
9: \( \text{success} \leftarrow \text{schedulePlan}(\mathcal{J}_k, S_{\text{temp}}, \mathcal{E}_{\text{temp}}) \)
10: if \( \text{success} \) then
11: \( it = \sum(t(e) - t(\text{pred}_e)), \forall \mathcal{J}_k \in S(e) \)
12: \( \text{minIdleTime} \leftarrow it \)
13: \( \text{bestPlan} \leftarrow \mathcal{J}_k \)
14: end if
15: end for
16: \( \text{Remove}(\mathcal{J}_k, V) \)
17: end if
18: end while
19: \( \text{return} \ (\mathcal{U}) \)

Algorithm 6 CheckConstraints

Require: A task \( J_i^k \), a starting time \( t(e) \) for \( J_i^k \)

Ensure: true if \( J_i^k \) satisfies the temporal constraints, false otherwise.
1: \( \{t(e) \in [r_i^k, d_i^k]\} \land \{t(e) \in [W_s, W_e]\} \land \{t(e) + p_i^k \in [W_s, W_e]\} \) then
2: \( \text{return true} \)
3: end if
4: \( \text{return false} \)

from a feasible schedule containing two plans \( \mathcal{P}_1 \) and \( \mathcal{P}_2 \), and we try to schedule in order the plans \( \mathcal{P}_3, \mathcal{P}_4 \) and \( \mathcal{P}_5 \). In Table IV are listed the parameters of the plans of tasks \( \mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_4, \mathcal{P}_5\} \).

Let \( \mathcal{P}_1 \) be the next plan to be scheduled, which has only one task \( J_i^2 \). The earliest starting time \( s_i^1 \) is set to the release time \( r_i^2 = 1 \). The event \( e_1 \in \mathcal{E} \mathcal{L} \) has a time instant \( t(e) = r_i^2 = 1 \), thus \( e_1 \) is the first event in which \( J_i^2 \) can be scheduled. The resource constraints in \( e_1 \) are satisfied for \( J_i^1 \), so it can be scheduled at \( t(e_1) = 2 \). The resource usages \( b_2(e_1) \) and \( b_3(e_2) \) are finally updated.

Table V shows the resulting Gantt diagram and the event list \( \mathcal{E} \mathcal{L} \) after the insertion of the plan \( \mathcal{P}_4 \).

Following, plan \( \mathcal{P}_4 \) is the next plan to be scheduled. The earliest starting time \( s_1^4 \) for \( J_i^4 \) is \( s_1^4 = t(e_1) = W_s \). At \( e_1 \) the temporal and resource constraints for \( J_i^4 \) are satisfied, so it is scheduled at \( s_1^4 = t(e_1) = 2 \), and then the resource usages \( b_4(e_1) \) is updated. Following, an insertion point is searched for the task \( J_i^3 \). The earliest starting time for \( J_i^3 \) is \( s_1^3 < t(e_1) = 6 \), so there is a time lag between \( J_i^3 \) and its predecessor \( J_i^2 \). The event \( e_4 \) is a good insertion point for the task \( J_i^3 \), since here the precedence, resource and temporal constraints are met. \( J_i^3 \) is finally scheduled at \( s_1^3 = t(e_4) = 6 \). The resource usage \( b_5(e_4) \) is updated. Since a feasible insertion point has been found for both \( J_i^3 \) and \( J_i^3 \), the plan \( \mathcal{P}_4 \) is scheduled correctly and the schedule generated is considered as feasible schedule.

Table VI shows the resulting Gantt diagram and the event list \( \mathcal{E} \mathcal{L} \) after the insertion of the plan \( \mathcal{P}_3 \). Finally, plan \( \mathcal{P}_5 \) is the last plan to be scheduled. The earliest starting time for \( J_i^5 \) is \( s_1^5 < 5 \). At \( t(e_5) = 5 \) the temporal and resource constraints for \( J_i^5 \) are not satisfied (\( b_2(e_5) = 1 \)). The next event \( e_6 \) is a good insertion point for the task \( J_i^5 \), since here the resource constraint and also the temporal constraints are met. The task \( J_i^5 \) is finally scheduled at \( s_1^5 = t(e_6) \). Following, an insertion point is searched for the task \( J_i^5 \), which consumes both the resources 1 and 3. The earliest starting time
for $J_2$ is $s_2^0 = 5$. The resources constraints are not satisfied in the events $e_3$, $e_4$ and $e_5$. The event $e_6$ is the first feasible insertion point for the task $J_2$, since at $t(e_6)$ the precedence and temporal constraints are met. Also, in $e_6$ both resources 1 and 3 are available ($b_1(e_6) = 0$ and $b_3(e_6) = 0$). The task $J_2$ is finally scheduled at $s_2^3 = t(e_6)$, and the usages of resources 1 and 3, respectively $b_1(e_6)$ and $b_3(e_6)$, are updated. Since a feasible insertion point has been found for both $J_1$ and $J_2$, the plan $Π_3$ is scheduled correctly and the schedule generated is considered as feasible schedule.

Table VII shows the resulting Gantt diagram and the event list $ℒ$ after the insertion of the plan $Π_3$.

### VI. COMPUTATIONAL RESULTS

In this section we present the computational results for the proposed heuristic. All the experiments were run on an entry-level machine (Intel i5-4210U, 6GB RAM, Linux OS). The algorithms were entirely written in Java (version 8).

We run different experiments of our algorithm by using a benchmark that depicts a realist scenario of a mission carried out by a Remote Piloted Air Vehicle (RPAS), which set of plans, listed in Table VIII, has to be scheduled. The validity of the mission is constrained between 0 and 180 seconds. Therefore, the time window $[W_s, W_e]$ is fixed to $[0, 180]$. We first evaluated the proposed heuristic on the benchmark listed in Table VIII. Then, we evaluated the performance of the proposed heuristic on different benchmarks obtained by modifying the set of plans listed in Table VIII. These modified benchmarks are opportune set up in such a way that the constraints are more or less relaxed. Thus, we studied eight different scenarios, which results are listed in Table IX. Following are listed the differences between each scenario:

- **Scenario 1**: set of plans listed in Table VIII;
- **Scenario 2**: in this scenario all the plans have the same temporal window $[r_k, d_k]$ fixed to $[0, 180]$, and the plans are all tied by precedence relations, so that $∀i, j$, $Π_i < Π_j$;
- **Scenario 3**: in this scenario we modified the release time of each plan in such way to have the maximum number of events;
- **Scenario 4**: in this scenario we duplicated all the plans with priority 6 and 8;
- **Scenario 5**: in this scenario we duplicated all the plans with priority 1 and 3;
- **Scenario 6**: in this scenario we used the same benchmark as the scenario 1, but we used a time window $[W_s, W_e]$ fixed to $[0, 90]$;
- **Scenario 7**: in this scenario we used the same benchmark as the scenario 1, but we used a time window $[W_s, W_e]$ fixed to $[0, 270]$;
- **Scenario 8**: in this scenario we used the same benchmark as the scenario 1, but each plan has a duplicate.

Table IX presents the execution times for each scenario. Each execution time is calculated as the mean of the execution times of ten runs of the algorithm on the same benchmark. The obtained computational results of the algorithm gave us a first proof of concept concerning the complexity of the scheduling problem we address. Moreover, the results are yet satisfactory in the context of scheduling plans for RPAS. As the problem of scheduling plans of tasks is difficult due to its
tight constraints, we conclude that the quality of the schedules provided by the proposed heuristic is good with respect to the complexity of the evaluated benchmarks.

### TABLE VII: Insertion of the plan $\Pi_5$ into a feasible schedule

| $\Pi_5$ | $n_k$ | $\alpha_k$ | $R^k$ | $r^k$, $d^k$ | $p^1_i$, $p^2_i$, $p^3_i$, $p^4_i$, $d^3_i$ | $\Pi_5 < \Pi_j$ | $J^1_i < J^2_i$ |
|---------|-------|------------|-------|-------------|---------------------------------|-----------------|-----------------|
| $\Pi_1$ | 2     | 3          | $(1,2)$ | 0, 180      | 3, 3                            | $J^1 < J^2$     | $J^1 < J^2$     |
| $\Pi_2$ | 2     | 3          | $(3,4)$ | 0, 180      | 4, 4                            |                 |                 |
| $\Pi_3$ | 1     | 3          | $(5)$   | 0, 180      | 3, 3                            |                 |                 |
| $\Pi_4$ | 1     | 3          | $(6)$   | 0, 180      | 3, 3                            |                 |                 |
| $\Pi_5$ | 1     | 3          | $(7)$   | 0, 180      | 3, 3                            |                 |                 |
| $\Pi_6$ | 1     | 3          | $(8)$   | 0, 180      | 3, 3                            |                 |                 |
| $\Pi_7$ | 4     | 1          | $(5,9,10,11)$ | 80, 160 | 20, 20, 20, 20 | $J^1 < J^2$     |                 |
| $\Pi_8$ | 4     | 1          | $(5,9,10,11)$ | 40, 120 | 20, 20, 20, 20 | $J^1 < J^2$     |                 |
| $\Pi_9$ | 4     | 1          | $(5,9,10,11)$ | 40, 120 | 20, 20, 20, 20 | $J^1 < J^2$     |                 |
| $\Pi_{10}$ | 3 | 5          | $(1,2,10)$ | 50, 80 | 5, 5, 5           | $J^1 < J^2$     | $J^1 < J^2$     |
| $\Pi_{11}$ | 4 | 5          | $(5,9,10,11)$ | 80, 120 | 20, 20, 20, 20 | $\Pi_{10} < \Pi_{11}$ | $J^1 < J^2$     |
| $\Pi_{12}$ | 4 | 5          | $(5,9,10,11)$ | 80, 120 | 20, 20, 20, 20 | $\Pi_{10} < \Pi_{11}$ | $J^1 < J^2$     |
| $\Pi_{13}$ | 3 | 5          | $(1,2,10)$ | 30, 180 | 5, 5, 5           | $\Pi_{13} < \Pi_{14}$ | $J^1 < J^2$     |
| $\Pi_{14}$ | 3 | 5          | $(1,2,10)$ | 30, 180 | 5, 5, 5           | $\Pi_{13} < \Pi_{14}$ | $J^1 < J^2$     |
| $\Pi_{15}$ | 3 | 6          | $(1,2,10)$ | 0, 20 | 5, 5, 5           | $\Pi_{20} < \Pi_{21}$ | $J^1 < J^2$     |
| $\Pi_{16}$ | 4 | 6          | $(5,9,10,11)$ | 20, 80 | 20, 20, 20, 20 | $\Pi_{15} < \Pi_{16}$ | $J^1 < J^2$     |
| $\Pi_{17}$ | 2 | 8          | $(3,4)$ | 60, 120 | 2, 2               | $J^1 < J^2$     |                 |
| $\Pi_{18}$ | 2 | 8          | $(3,4)$ | 60, 120 | 2, 2               | $J^1 < J^2$     |                 |
| $\Pi_{19}$ | 2 | 8          | $(3,4)$ | 60, 120 | 2, 2               | $J^1 < J^2$     |                 |
| $\Pi_{20}$ | 3 | 6          | $(1,2,10)$ | 0, 20 | 5, 5, 5           | $\Pi_{20} < \Pi_{21}$ | $J^1 < J^2$     |
| $\Pi_{21}$ | 4 | 6          | $(6,9,10,11)$ | 20, 60 | 20, 20, 20, 20 | $\Pi_{20} < \Pi_{21}$ | $J^1 < J^2$     |
| $\Pi_{22}$ | 2 | 6          | $(14,10)$ | 30, 70 | 2, 2               | $\Pi_{20} < \Pi_{21}$ | $J^1 < J^2$     |
| $\Pi_{23}$ | 4 | 1          | $(6,9,10,11)$ | 40, 90 | 20, 20, 20, 20 | $J^1 < J^2$     |                 |
| $\Pi_{24}$ | 4 | 1          | $(6,9,10,11)$ | 80, 150 | 20, 20, 20, 20 | $J^1 < J^2$     |                 |
| $\Pi_{25}$ | 4 | 1          | $(6,9,10,11)$ | 80, 130 | 20, 20, 20, 20 | $J^1 < J^2$     |                 |
| $\Pi_{26}$ | 2 | 4          | $(14,10)$ | 0, 180 | 2, 2               | $J^1 < J^2$     |                 |
| $\Pi_{27}$ | 3 | 6          | $(1,2,10)$ | 80, 120 | 5, 5, 5           | $J^1 < J^2$     |                 |
| $\Pi_{28}$ | 2 | 6          | $(3,4)$ | 120, 140 | 2, 2               | $\Pi_{28} < \Pi_{29}$ | $J^1 < J^2$     |
| $\Pi_{29}$ | 2 | 6          | $(14,10)$ | 120, 140 | 2, 2               | $\Pi_{28} < \Pi_{29}$ | $J^1 < J^2$     |
| $\Pi_{30}$ | 4 | 1          | $(6,9,10,11)$ | 150, 190 | 20, 20, 20, 20 | $\Pi_{28} < \Pi_{29}$ | $J^1 < J^2$     |
| $\Pi_{31}$ | 2 | 4          | $(3,4)$ | 120, 140 | 2, 2               | $\Pi_{28} < \Pi_{29}$ | $J^1 < J^2$     |
| $\Pi_{32}$ | 2 | 4          | $(3,4)$ | 120, 140 | 2, 2               | $\Pi_{28} < \Pi_{29}$ | $J^1 < J^2$     |

### TABLE VIII: The benchmark used for evaluating the performance of the proposed method.

| $K$ = $|P|$ | $K_{nk}$ | $K_{P_1}$ | $K_{P_2}$ | $|P_n|$ | Time (in ms) |
|-----------|---------|-----------|-----------|--------|-------------|
| Scenario 1 | 32      | 91        | 23        | 56     | $\sim 183$ |
| Scenario 2 | 32      | 91        | 23        | 67     | $\sim 185$ |
| Scenario 3 | 32      | 91        | 23        | 64     | $\sim 219$ |
| Scenario 4 | 42      | 118       | 30        | 68     | $\sim 316$ |
| Scenario 5 | 47      | 135       | 29        | 64     | $\sim 345$ |
| Scenario 6 | 32      | 91        | 16        | 35     | $\sim 197$ |
| Scenario 7 | 32      | 91        | 23        | 60     | $\sim 228$ |
| Scenario 8 | 64      | 182       | 37        | 81     | $\sim 558$ |

### TABLE IX: Execution times of the eight scenarios.

### VII. Future Works

#### A. Variable resource availability

The proposed heuristic considers only resources with a fixed availability of $B_\rho = 1, \forall \rho \in R$. A generalization of the algorithm could take into account resources with variable availability values, so that one resource could be used by different tasks at the same time. Augmenting the resource availability could also lead to better schedules in terms of number of plans scheduled into a fixed time window, as showed in Example 4.

**Example 4.** Let $\mathcal{P} = \{\Pi_1, \Pi_2\}$ be a set of two plans $\Pi_1$ and $\Pi_2$ which have each one task, respectively $J_1^1$ ($r^1_1 = 1, d^1_1 = 6, p^1_1 = 3$) and $J_2^2$ ($r^2_1 = 4, d^2_1 = 10, p^2_1 = 4$). In Figure 6 are...
showed two possible schedules for the task $J_1$. If the resource 1 has a maximum availability $B_1 = 1$ (Figure 6a), the earliest starting time for the task $J_1$ would be 6, because otherwise the usage of resource would exceed its availability. Instead, if the availability of the resource is 2 (Figure 6b), the earliest starting time would be set to 4 (that is, the release time of the task), and so the use of the resource is maximized. Also, in the schedule showed in Figure 6b the time interval $[5, 8]$ can be employed for scheduling other tasks, whereas this is not possible in the schedule in Figure 6a.

Fig. 6: Scheduling of task $J_1$, which has a release time of 4, into a resource with availability $B_1 = 1$ (6a) and a resource with availability $B_1 = 2$ (6b).

B. $p_k^i$ as function of time

The value of the processing time $p_k^i$ for a task $J_k^i$ could be modeled as a function of time $p_k^i(t)$ in such a way that the processing time of a task $J_k^i$ can vary according to the time instant in which it is scheduled. To motivate this extension, we present in Example 5 a case study of a RPAS that has to accomplish some in-flight operation.

**Example 5.** Let us consider a RPAS that has to accomplish a mission where it has to follow a straight trajectory and collect data from an object $R$ located in the terrain during a fixed time range, as depicted in Figure 7. During the time range in which the RPAS is within the object’s range it has to send information to the object $R$ in the terrain. The operation of sending information could require more time when the RPAS is far from the object, but less time when it is near the object. Thus, the processing time function related to the operation of sending information could be modeled as an inverse Gaussian function, as showed in Figure 8, where $d_{\text{max}}$ and $d_{\text{min}}$ are respectively the maximum and the minimum distance between the RPAS and the object $R$.

C. Alternative schedules

The proposed heuristic works in an incremental fashion, adding plans in order to a working schedule $S_w$ and keeping

![Fig. 7](image7.png)  
*Fig. 7: An example of RPAS that has to communicate informations to an object $R$. The processing time of the tasks communicating with $R$ could vary according to the distance between the object $R$ and the RPAS with respect to the time instant $t$."

![Fig. 8](image8.png)  
*Fig. 8: An inverse Gaussian used to model the processing time $p(t)$ of the the operation of sending information to a fixed object for the RPAS in Figure 7."
a set $\mathcal{P}_s$ of successfully scheduled plans, as well as a set $\mathcal{P}_f$ of unscheduled plans. The incremental process of construction of a feasible schedule can be represented as an directed acyclic graph, as shown in the example graph in Figure 9. According to this representation, each vertex represents a feasible schedule, while a directed arc $(i, j)$, which has label $\Pi_k$, represents the insertion of the plan $\Pi_k$ into the schedule $S_i$, that leads to a new feasible schedule $S_j$.

Fig. 9: The graph representation of a feasible schedule with four plans (in blue), and two possible alternative schedule (yellow and green).

The incremental manner of constructing the schedules has the advantage of being able to return a feasible schedule at each iteration of the algorithm.

Since the proposed method builds only a graph with no branches (no alternative schedule paths), a possible extension could consider different metrics to build alternative paths in the graph of solutions. For example, as the proposed heuristic returns a set $\mathcal{P}_f$ of plans unscheduled due to constraints violations, one possible extension could increase the priority values of the unscheduled plans in $\mathcal{P}_f$ while lowering those of a subset of feasible plans in $\mathcal{P}_s$. In this manner, alternative paths in the solution graph could be found. Also, these alternative paths could maximize further the objective function, thus allowing the heuristic method to escape from the local optima.

VIII. Conclusions
This article presented a heuristic algorithm for the problem of scheduling plans of tasks. The algorithm may be employed in realistic scenarios where the classical scheduling technique can not be used due to the considered constraints.

REFERENCES
[1] M. L. Pinedo, Scheduling. Springer US, 2012. DOI: 10.1007/978-1-4614-2361-4.
[2] R. Klein, Scheduling of Resource-Constrained Projects, vol. 10 of Operations Research/Computer Science Interfaces Series. Boston, MA: Springer US, 2000. DOI: 10.1007/978-1-4615-4629-0.
[3] L. Grivault, A. E. Fallah-Seghrouchni, and R. Girard-Claudon, “Agent-Based Architecture for Multi-sensors System Deployed on Airborne Platform,” pp. 86–89, IEEE, Sept. 2016.
[4] S. Kerkmennia and M. Nouvel-Fiani, “Toward common radar & EW multifunction active arrays,” pp. 777–784, IEEE, Oct. 2010.
[5] S. Imai, E. Blasch, A. Galli, W. Zhu, F. Lee, and C. A. Varela, “Airplane flight safety using error-tolerant data stream processing,” IEEE Aerospace and Electronic Systems Magazine, vol. 32, pp. 4–17, Apr. 2017.
[6] C. Schwindt and J. Zimmermann, eds., Handbook on Project Management and Scheduling Vol.1. Cham: Springer International Publishing, 2015. DOI: 10.1007/978-3-319-05443-8.
[7] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, Introduction to Algorithms. McGraw-Hill Higher Education, 2nd ed., 2001.
[8] A. B. Kahn, “Topological sorting of large networks,” Communications of the ACM, vol. 5, pp. 558–562, Nov, 1962.
[9] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan, “Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey,” Annals of Discrete Mathematics, vol. 5, pp. 287–326, Jan. 1979.
[10] P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch, “Resource-constrained project scheduling: Notation, classification, models, and methods,” European Journal of Operational Research, vol. 112, pp. 3–41, Jan, 1999.
[11] C. Artigues, S. Demasse, and E. Neron, Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications. ISTE, 2007.
[12] T. Peters, “Timsort.” http://bugs.python.org/file4451/timsort.txt, 2002.