Heuristic and metaheuristic methods for the multi-skill project scheduling problem with partial preemption

Oliver Polo Mejia, Christian Artigues, Pierre Lopez, Lars Mönch, Virginie Basini

Intl. Trans. in Op. Res. xx (2021) 1–34

Heuristic and Metaheuristic Methods for the Multi-Skill Project Scheduling Problem with Partial Preemption

Oliver Polo-Mejía [a,c], Christian Artigues[a,*], Pierre Lopez [a], Lars Mönch [b] and Virginie Basini [c]

[a] *LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France*
[b] *Department of Mathematics and Computer Science, University of Hagen, Hagen, Germany*
[c] *CEA, DEN, DEC, SETC, St. Paul lez Durance, France*
Received DD MMMM YYYY; received in revised form DD MMMM YYYY; accepted DD MMMM YYYY

Abstract

The multi-skill project scheduling problem (MSPSP) has been first addressed in the scheduling community for more than 15 years. This paper deals with a new variant of this problem, the multi-skill project scheduling problem with partial preemption (MSPSP-PP), where only a subset of resources can be released during the preemption periods. Like the standard problem, this variant is NP-hard, because of that we propose in this article a series of heuristic algorithms to solve instances arising from an industrial application. First, we present a serial greedy algorithm, based on priority rules and a flow problem for resource allocation. To improve the solutions of the greedy algorithm, we then introduce a binary-tree-based search algorithm and a greedy randomised adaptive search procedure (GRASP). Finally, we propose a large neighbourhood search (LNS) algorithm integrating exact and heuristic methods. The best results in terms of solution quality and execution time are obtained by combining the GRASP algorithm and the LNS approach. Furthermore, the proposed GRASP algorithm is able to find new best results on 56 instances out of 216 on a standard MSPSP instance set which shows the quality of the approach even on special cases of the considered problem.

Keywords: multi-skill scheduling; partial preemption; GRASP; local search; LNS

1. Introduction and related work

In some industrial environments, such as nuclear research facilities described in Polo-Mejía et al. (2020) that inspired our study, scheduling activities need to take into account a set of skills and clearances required to execute the activities and to match them with those that some technicians master. In this context, where regulations require the presence of a group of technicians having a set of well-defined skills for the execution of an activity, the multi-skill project scheduling problem (MSPSP) is important, see for example Bellenguez-Morineau and Néron (2007); Bellenguez-Morineau (2008); Montoya et al. (2014);

* Author to whom all correspondence should be addressed (e-mail: artigues@laas.fr).

Correia and Saldanha-da Gama (2015); Almeida et al. (2016); Young et al. (2017); Almeida (2018); Almeida et al. (2019). The MSPSP, initially proposed to schedule IT development projects (Néron, 2002), turns out to be more challenging than traditional scheduling problems, due to additional decisions to be made: it is needed to decide not only which resources will be allocated to each activity, but also the skills with which they will contribute. The MSPSP is a generalisation of the well-known resource-constrained project scheduling problem (RCPSP) (Artigues, 2008), where resources are characterised by the skills they master, and non-preemptive tasks require a certain amount of resources with a specific skill. Determining a solution consists in computing the periods in which each activity is executed and also which resources are allocated to the activity at each period, while satisfying activity and resource constraints: a resource can execute only those skills it masters and must cover only one skill per activity. The MSPSP can also be seen as a multi-mode RCPSP (Noori and Taghizadeh, 2018) where each execution mode is defined for a feasible resource allocation. However, most of the time, the number of resulting modes can be prohibitive to use the conventional algorithms used for solving the multi-mode RCPSP.

As indicated in Bellenguez-Morineau (2006), the MSPSP is NP-hard, thus solving industrial-sized instances of the MSPSP is time-consuming. Exact methods are proposed, among them branch-and-bound (Bellenguez-Morineau and Néron, 2007), branch-and-price (Montoya et al., 2014), constraint programming (Young et al., 2017) and mixed-integer linear programming (Correia et al., 2012; Almeida et al., 2019). Heuristic methods are then necessary to tackle this type of instances in short computing times. Most of the heuristics for the MSPSP in the literature are based on using priority rules. In her thesis work, Bellenguez-Morineau (2006) presents various greedy algorithms: one based on the Serial Generation Scheme (SGS) and two using the Parallel Generation Scheme (PGS), all of them using priority rules for determining the order in which activities are considered at each iteration of the heuristics. Almeida et al. (2016) also present a greedy algorithm using the PGS. However, this time, the authors propose a multi-pass version of the algorithm, where different priority lists for activities and different criticality functions for resource allocation are tested and the best solution found is kept. Almeida et al. (2018) propose a biased random-key genetic algorithm that significantly improved the heuristic of Almeida et al. (2016). In independently carried-out research efforts, Myszkowski et al. (2013) test different priority rules, the traditional ones and some more complex ones, with an SGS algorithm. They conclude that more complex rules do not always lead to better results. Because of their ease of development and the quality of the results obtained, the authors propose algorithms based on priority rule as an element of more elaborate metaheuristics. Later on, Myszkowski and Siemieński (2016) present a basic GRASP for the MSPSP. At each iteration of the algorithm, a feasible priority list (sequence) for activities is randomly generated, then a randomised greedy-based algorithm based on an SGS is used to perform the resource allocation. For the local search phase, a series of swapping moves are performed on the initial sequence, and the randomised greedy-based algorithm is used again to allocate the resources. More recently, Myszkowski et al. (2018) proposed a hybrid differential evolution and greedy algorithm. Lin et al. (2020) propose a genetic programming hyper-heuristic algorithm that outperforms the heuristics of Myszkowski and Siemieński (2016) and Myszkowski et al. (2018). More contributions related to heuristic and metaheuristic methods for the MSPSP can be found in Kolisch and Heimerl (2012) and Li and Womer (2009).

All above-described approaches for the MSPSP assume that interruption of activities is not allowed once they have started. However, operational and regulatory restrictions of sensitive units cannot guar-

antee the uninterrupted execution of all activities; a preemptive version of the MSPSP is thus required. There are very few works on preemptive MSPSP (Javanmard et al., 2017); most of the time, they consider that all resources can be released during the periods where an activity is preempted.

In addition to the very specific requirements of skills and clearances for ensuring their executions, some activities in the considered nuclear research facility have another important characteristic: they can be interrupted (preempted) and resumed later; however, during these preemption periods not all resources can be released due to safety reasons. This is the case, for example, for experimental activities requiring an inert atmosphere for their execution. In practice, one can stop these activities and allow the technicians and pieces of equipment to be used for other activities. However, safety constraints force to preserve continuously (even during the preemption periods) the inert atmosphere from the beginning until the end of the activity. No variant of the multi-skill scheduling problem had been proposed to model this behaviour before the work of Polo-Mejía et al. (2018b), which introduces the concept of partial preemption, and presents how this concept can be integrated into the MSPSP, leading to the MSPSP with partial preemption (MSPSP-PP). The main idea behind the partial preemption is to allow activities to be interrupted but releasing only a subset of resources during the preemption periods. A more precise description of the concept is presented in Section 2, where we describe the problem at hand.

The MSPSP-PP, in the same way as the standard MSPSP, is NP-hard (Polo-Mejía et al., 2018a), and the use of exact methods can be prohibitive to solve large-sized industrial instances, as shown in Polo-Mejía et al. (2020). The industrial applications of scheduling models require, most of the time, to be able to obtain high-quality solution in limited time. That is why we propose in this article various heuristic and metaheuristic methods for the MSPSP-PP. We present in Section 3 a basic serial greedy algorithm, inspired by the work of Bellenguez-Morineau (2006), that will be the base of the other heuristics proposed in this paper. The algorithm makes extensive use of priority rules for generating the schedule and of the solution of flow problems for the allocation of the technicians. Following the proposition of Myszkowski et al. (2013), we take the basic priority list-based heuristic presented in Section 3 and use it within a modified version of the limited discrepancy search procedure introduced in Harvey and Ginsberg (1995), to propose in Section 4 a local search procedure. To improve the results of our local search algorithm, we describe in Section 5 a greedy randomised adaptive search procedure (GRASP), combining the greedy and the local search algorithm. Most of the GRASP algorithms found in the literature for the MSPSP, such as the one presented by Myszkowski and Siemieński (2016), do not learn from the results of past iterations. The inclusion of learning from past iterations is part of the GRASP algorithm proposed in Section 5. Finally, we present in Section 6 a Large Neighbourhood Search (LNS) algorithm, a hybrid procedure combining exact (linear and constraint programming) and our heuristic methods. To assess the performance of the proposed methods, we describe in Section 7 extensive computational experiments. Concluding remarks are presented in Section 8.

Note that this paper is a considerably extended version of the works presented in Polo-Mejía et al. (2019a), where we introduced a greedy algorithm and an initial version of the tree-based local search algorithm, and in the extended abstract of Polo-Mejía et al. (2019b), where the GRASP was sketched on a pure conceptual level without providing any computational results. Here, we extend these previous papers by including tests with larger instances and new configurations for the local search algorithm, together with an improved GRASP and new LNS algorithms.

2. Problem description and formulation

The MSPSP-PP considers a set of activities $\mathcal{A} = \{1, \ldots, n\}$ partitioned into three subsets $\mathcal{A} = \mathcal{A}^P \cup \mathcal{A}^{NP} \cup \mathcal{A}^{PP}$, where $\mathcal{A}^P$ is the set of fully preemptive activities, $\mathcal{A}^{NP}$ is the set of non-preemptive activities and $\mathcal{A}^{PP}$ is the set of partially preemptive activities. We consider a discrete time horizon $\mathcal{T} = \{1, \ldots, T\}$ and a set $\mathcal{L} = \{1, \ldots, L\}$ of skills. We have a set $\mathcal{O}$ of disjunctive resources that represent multi-skilled operators/technicians. A parameter $A_{ot} \in \{0, 1\}$ indicates if at each time period $t \in \mathcal{T}$, an operator $o \in \mathcal{O}$ is present ($A_{ot} = 1$) or absent ($A_{ot} = 0$). A parameter $m_{ol} \in \{0, 1\}$ indicates whether resource/technician $o \in \mathcal{O}$ masters skill $l \in \mathcal{L}$ ($m_{kl} = 1$) or not ($m_{kl} = 0$). $\mathcal{R}$ is a set of renewable resources of cumulative type. Each resource $k \in \mathcal{R}$ has a discrete availability $B_{kt} \in \mathbb{N}$ for each time period $t \in \mathcal{T}$. Such resources typically represent geographical areas of limited capacities, teams of mono-skilled operators, inert atmosphere chambers, etc. Each activity $i \in \mathcal{A}$ is defined by a release date $r_i \in \mathbb{N}$, a deadline $d_i \in \mathbb{N}$, a processing time $p_i \in \mathbb{N}$, a demand $b_{ik} \in \mathbb{N}$ for each cumulative resource $k \in \mathcal{R}$ and a requirement $a_{il} \in \mathbb{N}$ for each skill $l \in \mathcal{L}$. In addition, an activity must be assigned to a minimum number of technicians q_i. Furthermore, for each partially preemptive activity $i \in \mathcal{A}^{PP}$ and each resource $k \in \mathcal{R}$ a parameter $\rho_{ik} \in \{0, 1\}$ indicates whether resource k can be released during preemption periods of activity i ($\rho_{ik} = 0$) or not ($\rho_{ik} = 1$). Finally, there is a set E of precedence constraints, where $(i, j) \in E$ means that activity j cannot start before activity i is completed.

The problem can now be formulated by a binary linear program, as the one proposed in Polo-Mejía et al. (2020). Variables $x_{it} \in \{0, 1\}$ indicate whether activity $i \in \mathcal{A}$ is in process at time $t \in \mathcal{T}$. Variables $y_{it} \in \{0, 1\}$ are equal to 1 if and only if activity i is being preempted at time $t \in \mathcal{T}$, meaning that there must be a period $\tau < t$ such that $x_{i\tau} = 1$ and a period $\tau' > t$ such that $x_{i\tau'} = 1$, while $x_{it} = 0$. Variables $z_{oit} \in \{0, 1\}$ indicate whether activity $i \in \mathcal{A}$ is assigned to technician $o \in \mathcal{O}$ at time $t \in \mathcal{T}$ or not. Auxiliary variables are necessary to linearly express some constraints: $x^-_{it} \in \{0, 1\}$ is equal to 1 if activity i starts at a period $t' < t$; $x^+_{it} \in \{0, 1\}$ is equal to 1 if activity i ends at a period $t' > t$. For non-preemptive activities, the binary variable $Z_{oi} \in \{0, 1\}$ indicates that operator $o \in \mathcal{O}$ is allocated to activity i throughout its execution. The set of operators allocated to a non-preemptive activity cannot change during the activity process for safety and security reasons regarding the nuclear context. For preemptive and partially activities the set of allocated operators can be changed at any time.

Given these variables, the binary formulation (1)–(14) formally defines the problem.

$$\min C_{\max} \tag{1}$$

$$\left(\sum_{i \in \mathcal{A}} x_{it} + \sum_{i \in \mathcal{A}^{PP}} \rho_{ik} y_{it} \right) b_{ik} \leq B_{kt} \qquad \forall k \in \mathcal{R}, \forall t \in \mathcal{T} \tag{2}$$

$$\sum_{i \in \mathcal{A}} z_{oit} \leq A_{ot} \qquad \forall o \in \mathcal{O}, \forall t \in \mathcal{T} \tag{3}$$

$$a_{il} x_{it} \leq \sum_{o \in \mathcal{O}} m_{ol} z_{oit} \qquad \forall i \in \mathcal{A}, \forall l \in \mathcal{L}, \forall t \in \mathcal{T} \tag{4}$$

$$\sum_{o \in \mathcal{O}} z_{oit} \geq q_i x_{it} \qquad \forall i \in \mathcal{A}, \forall t \in \mathcal{T} \tag{5}$$

$$\sum_{t=r_i}^{d_i} x_{it} \geq p_i \qquad \forall i \in \mathcal{A} \tag{6}$$

$$p_i(1 - x_{jt}) \geq \sum_{t'=t}^{T} x_{it'} \qquad \forall (i,j) \in E, \forall t \in \mathcal{T} \tag{7}$$

$$x_{it}^{-} \geq x_{it'} \qquad \forall i \in \mathcal{A}^{NP} \cup \mathcal{A}^{PP}, \forall t \in \mathcal{T}, \forall t' \leq t \tag{8}$$

$$x_{it}^{+} \geq x_{it'} \qquad \forall i \in \mathcal{A}^{NP} \cup \mathcal{A}^{PP}, \forall t \in \mathcal{T}, \forall t' \geq t \tag{9}$$

$$y_{it} = x_{it}^{-} + x_{it}^{+} - x_{it} - 1 \qquad \forall i \in \mathcal{A}^{PP}, \forall t \in \mathcal{T} \tag{10}$$

$$x_{it}^{-} + x_{it}^{+} - x_{it} = 1 \qquad \forall i \in \mathcal{A}^{NP}, \forall t \in \mathcal{T} \tag{11}$$

$$z_{oit} \geq Z_{oi} + x_{it} - 1 \qquad \forall i \in \mathcal{A}^{NP}, \forall o \in \mathcal{O}, \forall t \in \mathcal{T} \tag{12}$$

$$z_{oit} \leq Z_{oi} \qquad \forall i \in \mathcal{A}^{NP}, \forall o \in \mathcal{O}, \forall t \in \mathcal{T} \tag{13}$$

$$C_{\max} \geq t x_{it} \qquad \forall i \in \mathcal{A}, \forall t \in \mathcal{T} \tag{14}$$

The objective (1) is to minimise the makespan. Constraints (2) ensure that the total demand for a resource $k \in \mathcal{R}$ at a given time $t \in \mathcal{T}$ does not exceed its capacity. An activity i occupies b_{ik} units of resource k at time t if it is in process at time t ($x_{it} = 1$) or if it is a partially preemptive activity $i \in \mathcal{A}^{PP}$ being preempted at time t ($y_{it} = 1$) and if the resource cannot be released for this activity ($\rho_{ik} = 1$). Constraints (3) ensure that at most one activity $i \in \mathcal{A}$ is assigned to operator $o \in \mathcal{O}$ at time $t \in \mathcal{T}$ if the operator is present ($A_{ot} = 1$). No activity can be assigned if the operator is absent ($A_{ot} = 0$). Constraints (4) ensure that an activity in process ($x_{it} = 1$) must be assigned to a_{il} operators mastering skill $l \in \mathcal{L}$. Constraints (5) ensure the fulfilment of the minimum number of allocated operators q_i to an activity $i \in \mathcal{A}$ in process at any time $t \in \mathcal{T}$. Constraints (6) enforce each activity to be in process during p_i time units. Constraints (7) state that for each precedence constraint $(i,j) \in E$ activity j cannot be processed at any time t if activity i is not completed at time t. Constraints (8) enforce variable x_{it}^{-} to be equal to 1 if activity i is in process before t while Constraints (9) enforce variable x_{it}^{+} to be equal to 1 if activity i is in process after t. Then Constraints (10) state that a partially preemptive activity is preempted if it is not in process at time t while being in process at periods before and after t. Constraints (11) specify that a non-preemptive activity must be processed at time t if it is in process before and after t. Finally, for each non-preemptive activity, Constraints (12) state that an operator o allocated to an activity i ($Z_{oi} = 1$) must remain allocated to an activity during all time periods in which i is in process. Constraints (13) prevent an operator not globally allocated to a non-preemptive activity ($Z_{oi} = 0$) from being allocated to the activity at any time period. Finally, Constraints (14) ensure that the makespan is not smaller than the completion time of every activity.

From these constraints, it is important to note that a preemptive activity $i \in \mathcal{A}^{P}$ can be preempted at any time, releasing all its allocated resources. A partially preemptive activity $i \in \mathcal{A}^{PP}$ can be also preempted at any time but only its allocated resources $k \in \mathcal{R}$ such that $\rho_{ik} = 0$ are released. For preemptive and partially preemptive activities, the technicians can be changed at any time regardless of whether the activity is preempted or not. However a non-preemptive activity must keep the same set of allocated technicians all along its execution.

Another feature that distinguishes our problem from the standard MSPSP, even for non-preemptive activities, is the fact that Constraints (4) ensure that for each activity and at any time of its execution the required number of allocated operators mastering each required skill is obtained. However, there is no constraint that prevents an operator from being employed for several skills simultaneously.

An example of an MSPSP-PP instance is shown in Appendix A; in particular, it illustrates the distinctive characteristics of our problem. This example is used as a running example throughout the paper.

3. Greedy Algorithm: Serial schedule generation scheme and operator allocation

The MSPSP-PP can be decomposed into two interrelated subproblems, namely an activity scheduling problem coupled to a resource allocation problem. We propose a greedy algorithm using a serial Schedule Generation Scheme (SGS) that iteratively selects an activity for being scheduled with a priority rule and determines the earliest resource- and precedence-feasible start time. By solving a Minimum-Cost Maximum-Flow (MCMF) problem (Ababei and Kavasseri, 2010) for the selected activity, the operator allocation is further optimised. We first describe the greedy serial schedule generation scheme in Section 3.1, followed by the MCMF operator allocation procedure in Section 3.2. The assessed priority rules and a multi-pass variant of the algorithm are presented in Section 3.3.

3.1. The serial schedule generation scheme

All the material presented in this section is a considerably extended version of the conference paper by Polo-Mejía et al. (2019a), except that, to make the research reproducible, much more details are provided here on the scheduling and resource allocation procedures. The algorithm is a direct extension of the serial SGS initially proposed for the resource-constrained project scheduling problem in Kolisch (1996). The main differences are the possibility of preemption and partial preemption and the management of operators and skills.

Regardless of whether the activity is preemptive or not, the schedule of an activity is represented by a set of time points $\mathcal{T}_i$, which are stored as an ordered list of non-overlapping discrete intervals. The smallest time period in $\mathcal{T}_i$ is the activity start time S_i and the largest time period in $\mathcal{T}_i$ is the activity completion time C_i. For a non-preemptive activity, there is a single interval in $\mathcal{T}_i$ with $\mathcal{T}_i = [\![S_i, C_i = S_i + p_i - 1]\!]$.[1] For a (partially) preemptive activity that has been preempted ν_i times, there is a set of intervals such that $\mathcal{T}_i = [\![S_i, t_1]\!] \cup [\![t_1, t_2]\!] \cup \ldots \cup [\![t_{\nu_i-1}, C_i]\!]$. Note that there is no interest of preempting or changing the operator allocation of an activity at a time period over which no activity starts, ends, or no operator or resource becomes unavailable (due to initial availabilities A_{ot} and B_{kt}). Hence the number of execution intervals in $\mathcal{T}_i$ is independent of the time horizon T as it is lower than $\min(p_i, 2n + Q)$ where Q is the total number of different breakpoints of the piece-wise constant initial availability functions B_{kt} and A_{ot}.

The serial SGS (Algorithm 1) is a greedy constructive algorithms that starts with an empty schedule,

[1] Note that the use of notation $[\![.]\!]$ refers to discrete intervals.

selects an activity at each iteration from a list $\mathcal{Q}$ initialised to the set of activities at line 1, and schedules it at its earliest precedence- and resource-feasible starting time taking account of the previously scheduled activities (lines 3–38). To evaluate resource feasibility the algorithm maintains a resource timetable $\overline{\mathcal{I}}$ in the form of a set of discrete intervals that partition the time horizon $\mathcal{T}$ such that the resource and operator availability is constant over each interval. The timetable is initialised with the intervals issued from the initial resource and operator availability A_{ot} and B_{kt} (line 2). For the same reason as explained above, the number of intervals in the timetable is lower than $2n + Q$.

An interval $\bar{I}_q$ of $\overline{\mathcal{I}}$ is defined by its left bound $\bar{t}_q$, while $\bar{t}_{q+1}$ refers to the left bound of the subsequent interval, therefore we have $\bar{I}_q = [\![\bar{t}_q, \bar{t}_{q+1} - 1]\!]$ and $\overline{A}_{oq}$ (respectively $\overline{B}_{kq}$) refers to the constant availability of operator o (resp. resource k) during interval $\overline{I}_q$.

The efficiency of the serial SGS lies in the fact that the execution periods in $\mathcal{T}_i$ of the selected activity is obtained by traversing in the worst case the intervals in $\overline{\mathcal{I}}$ in a single pass (of at most $2n + Q$ steps). We detail this process thereafter.

At each iteration, one activity i is extracted from $\mathcal{Q}$ (line 4), assuming in a conventional way that an activity cannot be selected if one of its predecessor is still in $\mathcal{Q}$. Then the earliest start time ES of the selected activity is determined at line 5 as the maximum of its release date and the largest completion time of its predecessors. The algorithm then determines the earliest set of time periods in $\mathcal{T}_i \in [\![ES, T]\!]$ over which the activity can be scheduled (lines 5–32). For that, an index q of the intervals in timetable $\overline{\mathcal{I}}$ is first positioned at the first interval such that $ES \in [\![\bar{t}_q, \bar{t}_{q+1} - 1]\!]$ (lines 5–8) to check the resource availability starting from time ES. The remaining duration P and the set of scheduling intervals $\mathcal{T}_i$ are initialised to the activity duration and the empty set (line 9). Then, lines 10–32 perform the traversal of intervals $[\![\bar{t}_q, \bar{t}_{q+1} - 1]\!]$ from $\overline{\mathcal{I}}$ while there remain unassigned time periods for the activity. We first compute at line 11 the set $\mathcal{O}_i^{temp}$ of operators available in interval $[\![\bar{t}_q, \bar{t}_{q+1} - 1]\!]$ that master at least one skill required by the activity. For a non-preemptive activity, the set of allocated operators must be the same during all the activity execution intervals. So, if i is a non-preemptive activity that already tentatively starts before the examined interval q, lines 12–16 update the set $\mathcal{O}_i$ of available operators from the start by taking the intersection with $\mathcal{O}_i^{temp}$. Otherwise, if the activity is preemptive, partially preemptive or not already started, only the operators available during the examined interval matter, therefore $\mathcal{O}_i$ is set to $\mathcal{O}_{iq}$.

If the set $\mathcal{O}_i$ contains at least q_i operators able to fulfil the requirement a_{il} of the activity for each skill $l \in \mathcal{L}$ and if all resources required by the activity are sufficiently available (test at line 17), then a portion of the activity can be scheduled on the considered interval. If the activity is not started yet (Boolean $tentStart$), the start time of the activity S_i is set to its earliest start time in the interval (line 19), the activity is marked as tentatively started (line 19). If i is non-preemptive, $\mathcal{T}_i$ is tentatively set to a single interval from S_i to $S_i + p_i - 1$ (line 21), otherwise, if the activity is preemptive or partially preemptive, there remain P time units to be scheduled. The activity will therefore occupy all time periods from $\tau = \max(S_i, t_q)$ until $\tau' = \min(\tau + P - 1, t_{q+1} - 1)$ in the current interval $[\![t_q, t_{q+1} - 1]\!]$, which allows the insertion of interval $[\![\tau, \tau']\!]$ in $\mathcal{T}_i$ (line 25). Then, the remaining duration P is decreased by the amount that can be placed in the interval.

If there is not enough available resource or operators for the activity in the examined interval, the

Algorithm 1: Serial SGS

$\mathcal{Q} \leftarrow \mathcal{A}$
Initialise $\overline{\mathcal{I}}$ by intervals issued from A_{ot} and B_{kt}, $\forall o \in \mathcal{O}, k \in \mathcal{R}, t \in \mathcal{T}$
while $\mathcal{Q} \neq \emptyset$ **do**
  Select an activity i from $\mathcal{Q}$ by a priority rule.
  $ES \leftarrow \max(r_i, \max_{(j,i)\in E} C_i)$; $q \leftarrow 1$
  while $\bar{t}_{q+1} \geq ES$ **do**
    $q \leftarrow q + 1$
  end
  $P \leftarrow p_i$; $\mathcal{T}_i \leftarrow \emptyset$; $tentStart \leftarrow false$
  while $P > 0$ **do**
    $\mathcal{O}_i^{temp} \leftarrow \{o \in \mathcal{O} | \bar{A}_{oq} = 1 \text{ and } \exists l \in \mathcal{L}, a_{il} \geq m_{ol} \geq 1\}$
    if $i \in \mathcal{A}^{NP}$ *and* $tentStart = true$ **then**
      $\mathcal{O}_i \leftarrow \mathcal{O}_i \cap \mathcal{O}_i^{temp}$
    else
      $\mathcal{O}_i \leftarrow O_i^{temp}$
    end
    if $|\mathcal{O}_i| \geq q_i$ *and* $\forall l \in \mathcal{L}$, $\sum_{o\in\mathcal{O}_i} m_{ol} \geq a_{il}$ *and* $\forall k \in \mathcal{R}$, $\bar{B}_{kq} \geq b_{ik}$ **then**
      if $tentStart = false$ **then**
        $S_i \leftarrow \max(ES, \bar{t}_q)$; $tentStart \leftarrow true$
        if $i \in \mathcal{A}^{NP}$ **then**
          $\mathcal{T}_i \leftarrow [\![S_i, S_i + p_i - 1]\!]$
        end
      end
      if $i \notin \mathcal{A}^{NP}$ **then**
        $\mathcal{T}_i \leftarrow \mathcal{T}_i \cup [\![\max(S_i, \bar{t}_q), \min\left(\max(S_i, \bar{t}_q) + P - 1, \bar{t}_{q+1} - 1\right)]\!]$
      end
      $P \leftarrow \max\left(0, P - \bar{t}_{q+1} + \max(S_i, \bar{t}_q)\right)$
    else if $tentStart = true$ *and* ($i \in \mathcal{A}^{NP}$ *or* ($i \in \mathcal{A}^{PP}$ *and* $\exists k \in \mathcal{R}$, $\rho_{ik} = 1$, $\bar{B}_{kq} < b_{ik}$)) **then**
      $tentStart \leftarrow false$; $\mathcal{T}_i \leftarrow \emptyset$; $P \leftarrow p_i$
    end
    $q \leftarrow q + 1$
  end
  set $C_i = \max\{t' | [\![t, t']\!] \in \mathcal{T}_i\}$
  if $C_i > d_i$ **then**
    exit `// fail`
  end
  Obtain operator allocations $\mathcal{O}_{ir}, r = 1, \ldots, |\mathcal{T}_i|$ by solving the MCMF subproblem for each interval $I_r \in \mathcal{T}_i$.
  Update $\overline{\mathcal{I}}$ with intervals $\mathcal{T}_i$ and operator allocations $(\mathcal{O}_{ir})_{r=1..|\mathcal{T}_i|}$
  $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{i\}$
end

current tentative start at S_i must be aborted in all the cases for a non-preemptive activity. For a partially preemptive activity the start time at S_i must be aborted if the non-releasable resources are available in a non-sufficient amount (test at line 28). In such situations, the activity is marked as not started and the remaining time periods and set of scheduling intervals are reinitialised.

The intervals are enumerated by incrementing q (line 31) until the activity is fully scheduled, i.e. when $P = 0$. The completion time of the activity is then obtained from $\mathcal{T}_i$ (line 33). The greedy algorithm fails if the completion time exceeds the deadline. Otherwise, we proceed to operator allocation. Note that to obtain a feasible operator allocation, it is sufficient to greedily select a_{il} operators that master skill $l \in \mathcal{L}$. That requires $O(L|\mathcal{O}|)$ time. However, as the operator allocation highly influences the schedule quality, we decide for a smarter operator allocation. A minimum-cost maximum-flow problem is solved on each interval in $\mathcal{T}_i$ to optimise the operator allocation (line 37). The MCMF procedure outputs a set of allocated operators, where $\mathcal{O}_{iq}$ denotes the set of operators allocated to i for the q^{th} interval I_q in $\mathcal{T}_i$. The timetable $\overline{\mathcal{I}}$ is then updated using scheduled time periods $\mathcal{T}_i$, operator allocation $\mathcal{O}_{iq}$, and removing i from $\mathcal{Q}$.

Without solving the MCMF problems, the time complexity of the algorithm is $O(n(n+Q)(L|\mathcal{O}| + |\mathcal{R}|)$. The MCMF procedure and additional computational efforts are detailed in Section 3.2.

3.2. The minimum-cost maximum-flow operator allocation procedure

We aim at finding a feasible operator allocation that preserves future allocations w.r.t. operator scarceness. To that purpose, we compute a criticality indicator c_o of an operator o as follows. First, let g_{io} denote the correlation indicator that measures the degree to which activity i might require operator o for its execution:

$$g_{io} = |\{l \in \mathcal{L} | m_{ol} = 1\} \cap \{l \in \mathcal{L} | a_{il} > 0\}|.$$

Then, the criticality of operator o relatively to the activity i to be scheduled is equal to the following ratio c_o, where the numerator estimates the degree to which the operator should be allocated to the unscheduled activities except i while the numerator is the correlation indicator with i:

$$c_o = \frac{\sum_{j \in \mathcal{Q} \setminus \{i\}} p_j g_{jo}}{g_{io}}.$$

Finding the subset of operator of minimum criticality among a set $\mathcal{O}_i$ on a time range $[\![t, t']\!]$ can be approximated by solving an MCMF problem. Such an MCMF flow for operator allocation with criticality costs was proposed by Bellenguez-Morineau (2006) for the standard MSPSP where an operator can cover at most a single skill when allocated to an activity. We proposed a variant where the operator can cover several skills and with a minimum number of required operators for the activity. We create a source node 0 connected to one node per required skill l such that $a_{il} > 0$ with an arc of maximal capacity a_{il} and zero cost. We also create an arc connecting the source to a dummy skill $L + 1$ node (corresponding to the minimum required number of operators) of maximum capacity q_i and cost zero. We then create an arc between each skill node l (including the dummy one) and a node per operator $o \in \overline{\mathcal{O}}$ such that $m_{ol} = 1$ of maximal capacity 1 and 0 cost. Finally, we create a sink

node $*$ and we connect each operator node to the sink node with an infinite capacity and criticality cost c_o. The problem is then solved by solving a minimum cost flow of value $\sum_{l\in\mathcal{L}} a_{il} + q_i$. Let $L_i^* = \sum_{o\in\mathcal{O}_i} |\{l \in \mathcal{L}, m_{ol} = 1\}|$ be the number of arcs between skills and operator nodes and let $c^{\max}$ denote the maximum criticality cost. As there are $O(L_i + |\mathcal{O}_i|)$ nodes and $L_i + |\mathcal{O}_i| + L_i^*$ arcs, the MCMF problem can be solved in $O((L_i^*|\mathcal{O}_i|)^2(L_i + |\mathcal{O}_i| + L_i^+)\log((L_i + |\mathcal{O}_i|)c^{\max})$ by the cost scaling push-relabel algorithm (Goldberg, 1997).

See Appendix B for an illustration of an iteration of the greedy algorithm.

3.3. Priority rules and multi-pass variant

The greedy algorithm can be used with a single priority rule but better solutions can be obtained in a multi-pass version where the algorithm is launched successively with different priority rules and the schedule with the best $C_{\max}$ is the result.

For the multi-pass test, we use the most common priority rules in the scheduling literature (Panwalkar and Iskander, 1977): Longest Duration, Most Successors, Earliest Start Time, Earliest Finish Time, Greatest Rank Positional Weight, Greatest Resource Demand, Latest Start Time, and Minimum Slack.

Whatever the priority rule is, the algorithm first schedules all activities having a deadline, then all non-preemptive activities, then all partially preemptive activities, and finally all preemptive activities. The idea is to consider the most constrained activities first. Such single or multiple priority rule-based procedures may fail in obtaining a feasible solution because of the deadlines. This typically does not happen in the industrial problem as only a few activities have deadlines and they are not very restrictive. This is why our methods tends to scheduling activities with deadline first but have no means to enforce strict feasibility.

4. Tree-based Local Search Algorithm

Greedy construction algorithms are known by their risk of accepting myopic choices that may lead to local optima (Voß et al., 2005). Inspired by the limited discrepancy search proposed by Harvey and Ginsberg (1995), we propose to execute an additional phase in such procedures, based on a tree-based local search algorithm to improve an incumbent solution. For each sequence used in the generation phase, there is a large amount of possible schedules that are defined by the technician allocations we made. The fact of choosing a specific technician may change the earliest start time of future activities since this decision modifies technician availability, and thus producing different schedules. The objective of the tree-based local search algorithm is to visit some of these possible schedules.

4.1. Enumeration of alternative operator allocations via tree search

The tree-search algorithm maintains a set of open nodes $\mathcal{N}$ where each node represent a stage of the greedy SGS with a selected activity and a possible operator allocation.

More precisely, a node $N \in \mathcal{N}$ is a tuple storing the current set of unscheduled activities, the resource and operator timetable, the execution intervals of scheduled activities and their operator allocations, as well as a depth indicator Δ and the number of discrepancies $\#discr$ (selection of an alternative operator allocation instead of the preferred one) used in the branch so far, which gives respectively:

$$N = \left(\mathcal{Q}, \overline{\mathcal{I}}, (\mathcal{T}_i)_{i \in \mathcal{A} \setminus \mathcal{Q}}, (\mathcal{O}_{ir})_{i \in \mathcal{A} \setminus \mathcal{Q}, I_r \in \mathcal{T}_i}, \Delta, \#discr\right).$$

Let us come back to the operator allocation step (line 37) of Algorithm 1. If, for a given interval I_q in $\mathcal{T}_i$, the MCMF subproblem computes preferred operator allocation $\mathcal{O}_{ir}$, then alternative operator sets can be possibly obtained by solving $|\mathcal{O}_{ir}|$ MCMF problems, each being obtained by removing one operator of $\mathcal{O}_{ir}$ from the available operators. Let the best operator allocation obtained this way be denoted by $\mathcal{O}'_{ir}$ and consider it as the alternative allocation. For a non-preemptive task there are only two operator allocations, the preferred one and the alternative one, as $|\mathcal{T}_i| = 1$. But for a preemptive or partially preemptive activity we have in the worst case $|\mathcal{T}_i| = \min(p_i, 2n + Q)$, which yields $2^{|\mathcal{T}_i|}$ different operator allocations, i.e. child nodes of the current node.

Appendix C is illustrates the local search algorithm on the running example. Appendix C.1 shows the local search tree.

4.2. Controlling the search by probabilistic traversal of an equivalent binary tree

However, there is in general an exponential number of nodes; therefore, we introduce different parameters to control the search. The main parameter to control the search is the probability π to select the alternative operator allocation instead of the preferred one for each execution interval of an activity. This is the main parameter of the proposed algorithm since it controls the number of visited branches, and thus the time required to visit the generated tree, and the solution quality. It is also experimentally determined that, for the same π value, a higher portion of preemptive or partially preemptive activities in the instance increases the time required by the algorithm. To limit the computing time, we propose, after several preliminary experiments, to use a version of this parameter based on a function that depends on the number of non-preemptive activities present in the instance (i.e. self-adaptive to the instance characteristics):

$$\pi = \pi^{-} e^{\frac{\ln(\pi^{+}) - \ln(\pi^{-})}{|\mathcal{A}|} |\mathcal{A}^{NP}|},$$

where π^{-} and π^{+} are input parameters giving the minimum and the maximum values the initial probability can take.

The probability to accept a given alternative for one of the execution intervals of a selected activity is $\pi(1 - \Delta/\Delta^{\max})$ where Δ is the depth of the alternative allocation in a binary tree equivalent to the search tree presented in the previous section and $\Delta^{\max}$ is the maximal depth of this binary tree.

Furthermore, from the way schedules are generated, we can expect that the chance of making poor decisions by the heuristic decreases as we add more activities to the partial schedule (going deep in the search tree); if there are fewer activities to be scheduled, the criticality cost of a technician is more accurate. We exploit this characteristic by assigning to each alternative operator allocation a probability that depends on its depth in the equivalent binary search tree where each interval of a preemptive or

partially preemptive activity is considered as a non-preemptive task.

See Appendix C.2 for the binary search tree equivalent to the tree given in Appendix C.1.

4.3. Algorithm description

Algorithm 2 describes the tree based local search procedure. Line 3 pushes the root node in the node stack $\mathcal{N}$, with no activity scheduled and the initial resource timetable, empty scheduled intervals, operator allocation and an initial depth equal to 0. Then the local search begins while there remains unexplored nodes, which constitutes the main loop (lines 4–34). Following the depth-first search principle, the latest added node is taken from $\mathcal{N}$, giving the partial schedule made of unscheduled activities $\mathcal{Q}$, the operator and resource timetable $\overline{\mathcal{I}}$, the execution intervals for scheduled activities $(\mathcal{T}_j)_{j\in\mathcal{A}\setminus\mathcal{Q}}$, their operator allocations $(\mathcal{O}_{jr})_{j\in\mathcal{A}\setminus\mathcal{Q},I_r\in\mathcal{T}_i}$, and the node depth Δ (line 5). An elementary step of the greedy SGS is then applied by selecting an activity i from $\mathcal{Q}$ and obtaining its feasible earliest execution intervals $\mathcal{T}_i$ (lines 6-7). If the reachable $C_{\max}$ (which is either the obtained makespan if $\mathcal{Q}$ is empty or a simple lower bound based on the earliest completion time of the activities taking account of their possibly scheduled predecessors) is not better than the makespan of the best solution $C^*_{\max}$ (i.e. when test of line 8 fails) or if the completion time of i exceeds its deadline, the node is not developed further. Otherwise, the preferred and alternative operator allocations of each interval in $\mathcal{T}_i$ are computed by the MCMF (line 9). If all activities are scheduled, we reach a leaf node and the new best solution (schedule and operator allocations) is stored (line 12).

Otherwise, the different operator allocations for activity i are generated by traversing a local binary tree. To that purpose, we use a second node stack $\mathcal{B}$ in which a node is a tuple $(r, (\bar{\mathcal{O}}_{iq})_{q=1..r}, \#ldiscr)$ where r is the index of the last interval for which the operator allocation is fixed and $(\bar{\mathcal{O}}_{iq})_{q=1..r}$ is the selected preferred or alternative operator allocations for i up to interval I_r in the considered branch; $\#ldiscr$ is the number of discrepancies in the branch of the binary tree. When a node is taken from $\mathcal{B}$ (line 16), we first check if all intervals in $\mathcal{T}_i$ have an operator allocation (line 17), in which case we are at a leaf of the binary tree at depth $r = |\mathcal{T}_i|$. The resource timetable is then updated with the schedule and the obtained operator allocations of the execution of intervals i (line 18) and these operator allocations are added to the set of operator allocations of the scheduled activities (line 19). A child node in the global search tree is pushed to a temporary stack $\mathcal{N}^{temp}$. The node stores $\mathcal{Q}$, the updated timetable $\overline{\mathcal{I}}$, the schedule, the operator allocations and the depth (line 20). Otherwise, the node of binary tree does not correspond to a leaf and the next interval is considered by incrementing r. The alternative allocation (right child node) is only considered with a probability $\pi\,(1-(\Delta+r)/\Delta^{\max})$, as the node depth in the global binary tree would be $\Delta + r$. If the probability is reached by the random value and if a maximum number of discrepancies per branch $NBMAXDISCR$ is not reached, the alternative operator allocation for interval I_r is added to the set of operator allocations for execution intervals of the parent node (line 24) and a new node with the new interval index r, the set of operator allocations up to r and the updated cumulative probability is pushed to $\mathcal{B}$ (line 25). Then, the preferred operator allocation for interval I_r is added to the set of operator allocations for execution intervals of i of the parent node (line 27) and the new node is generated as for the right child but with an unchanged cumulative probability (line 28). Finally, when the binary tree has been explored, all new nodes stored in stack

Algorithm 2: Tree-based local search algorithm

$\mathcal{Q} \leftarrow \mathcal{A}$
Initialise $\overline{\mathcal{I}}$ by intervals issued from A_{ot} and B_{kt}, $\forall o \in \mathcal{O}, k \in \mathcal{R}, t \in \mathcal{T}$
$\mathcal{N} \leftarrow \{(\mathcal{Q}, \overline{\mathcal{I}}, \emptyset, \emptyset, 0, 0)\}$
while $\mathcal{N} \neq \emptyset$ **do**
    Pop node $\left(\mathcal{Q}, \overline{\mathcal{I}}, (\mathcal{T}_j)_{j \in \mathcal{A} \setminus \mathcal{Q}}, (\mathcal{O}_{jq})_{j \in \mathcal{A} \setminus \mathcal{Q}, I_q \in \mathcal{T}_i}, \Delta, \#discr\right)$ from $\mathcal{N}$
    Select an activity i from $\mathcal{Q}$ by a priority rule.
    Apply steps 5–32 of Algorithm 1
    if $C_i \leq d_i$ *and* $C_{\max} < C^*_{\max}$ **then**
        Obtain preferred and alternative operator allocations $\mathcal{O}_{ir}$ and $\mathcal{O}'_{ir}$, by solving the $|\mathcal{O}_{ir}|$ MCMF subproblems for each interval $I_r \in \mathcal{T}_i$.
        $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{i\}$
        if $\mathcal{Q} = \emptyset$ **then**
            store solution as the best one and update $C^*_{\max}$
        else
            $\mathcal{B} \leftarrow \{(0, \emptyset, 0)\}$; $\mathcal{N}^{temp} \leftarrow \emptyset$
            while $\mathcal{B} \neq \emptyset$ **do**
                pop $(r, (\bar{\mathcal{O}}_{iq})_{q=1..r}, \#ldiscr)$ from $\mathcal{B}$
                if $r = |\mathcal{T}_i|$ **then**
                    Update $\overline{I}$ with intervals $\mathcal{T}_i$ and operator allocationallocation $(\bar{\mathcal{O}}_{iq})_{q=1..r}$
                    $\mathcal{O}_{iq} \leftarrow \bar{\mathcal{O}}_{iq}, \forall q = 1, \ldots, r$
                    push node $\left(\mathcal{Q}, \overline{\mathcal{I}}, (\mathcal{T}_j)_{j \in \mathcal{A} \setminus \mathcal{Q}}, (\mathcal{O}_{jr})_{j \in \mathcal{A} \setminus \mathcal{Q}, r=1..q}, \Delta + r, \#discr + \#ldiscr\right)$ on $\mathcal{N}^{temp}$
                else
                    $r \leftarrow r + 1$
                    if *random(0,1)*$\leq \pi\left(1 - (\Delta + r)/\Delta^{\max}\right)$ *and* $\#discr + \#ldiscr + 1 < NBMAXDISCR$ **then**
                        $\bar{\mathcal{O}}_{ir} \leftarrow \mathcal{O}'_{ir}$
                        push $(r, (\bar{\mathcal{O}}_{iq})_{q=1..r}, \pi\left(1 - (\Delta + r)/\Delta^{\max}\right), \#ldiscr + 1)$ to $\mathcal{B}$
                    end
                    $\bar{\mathcal{O}}_{ir} \leftarrow \mathcal{O}_{ir}$
                    push $(r, (\bar{\mathcal{O}}_{iq})_{q=1..r}, \#ldiscr)$ on $\mathcal{B}$
                end
                push all nodes of $\mathcal{N}^{temp}$ to $\mathcal{N}$
            end
        end
    end
end

$\mathcal{N}^{temp}$ are popped from $\mathcal{N}^{temp}$ and pushed in $\mathcal{N}$ to reverse their order accordingly to the depth-first search (line 30).

See Appendix D for an illustration of an iteration of Algorithm 2.

5. Greedy randomised Adaptive Search Procedure

GRASP is an iterative multi-start algorithm in which each iteration consists of two phases: generation of a feasible solution (construction phase) and improvements by local search. Surveys of GRASP and its applications are presented in Resende and Ribeiro (2019); Festa and Resende (2009a,b). For the MSPSP-PP, in the construction phase, a feasible solution is generated by means of the greedy SGS (Algorithm 1) with an adaptive priority rule based on a restricted candidate list updated at each iteration of the SGS. The neighbourhood is then explored by the tree-based local search (Algorithm 2) until a local optimum is found. The best solution found over all GRASP iterations is kept as a result. Hereafter, we describe the GRASP algorithm we propose to solve the MSPSP-PP.

In Section 5.1 we describe how the restricted candidate list is built, based on an adaptive greedy evaluation function described precisely in Section 5.2, which includes priority, intensification and feasibility components. The intensification component is based on the storage of an elite solution set regularly updated, as explained in Section 5.3.

5.1. Restricted candidate list construction and activity selection

The GRASP algorithm specified by Algorithm 3, uses a specific activity selection procedure to extract the activity to be scheduled at line 4 of Algorithm 1. Let $F(i)$ be an adaptive greedy evaluation function which indicates the degree of relevance of selecting activity i in the current greedy iteration. Recall that $|\mathcal{Q}|$ is the number of not yet scheduled activities after the current greedy iteration. The restricted candidate list (RCL) is built at this step. It is made up of the $1 + \lfloor \alpha |\mathcal{Q}| \rfloor$, $\alpha \in [0, 1]$, activities having the best $F(i)$ values. Each activity within the RCL has a probability of being chosen (π_i) defined as follows:

$$\pi_i = \frac{F(i)}{\sum_{j \in RCL} F(j)} \, .$$

For choosing the value of α, we apply the reactive strategy proposed by Prais and Ribeiro (2000), where the value of α is randomly selected from a discrete set $\Psi = \{\alpha_1, ..., \alpha_n\}$ of possible α values. The probabilities associated with the choice of each value are all initially uniformly distributed. After a few iterations, they are periodically recalculated, taking into consideration the quality of the obtained solution for each $\alpha_k \in \Psi$. In our case, we use the $C_{\max}$ as a quality indicator. More precisely, for each $\alpha_k \in \Psi$, we compute the average makespan $\bar{C}_{\max}(\alpha_k)$ over the last N iterations and the sum $\chi = \sum_{\alpha_k \in \Psi} \frac{1}{\bar{C}_{\max}(\alpha_k)}$. To give a smaller probability to the α_k that yield larger $\bar{C}_{\max}(\alpha_k)$, we update the probability of α_k with $\frac{1}{\bar{C}_{\max}(\alpha_k)\chi}$ (so the sum of probabilities is 1).

Algorithm 3: GRASP for the MSPSP-PP

$git \leftarrow 0$; $fail \leftarrow 0$; $it \leftarrow 0$; $fit \leftarrow 0$; $\varepsilon \leftarrow \emptyset$; $OLDSIM \leftarrow \infty$; $OLDFAIL \leftarrow -\infty$
Choose randomly α and initialise β, δ and γ to 0.33;
while $git \leq MAXITER$ **do**
    $it \leftarrow it + 1$ `// Iterations counter for parameters`
    `// Generate initial solution`
    Run greedy SGS (algorithm 1) with parameters α, β, δ and γ to update the RCL
    if *Algorithm 1 found a feasible solution with sequence* σ **then**
        Run tree-based local search (Algorithm 2) using sequence σ for selecting activities;
        Update ε with σ;
        $git \leftarrow git + 1$
    else
        $fail \leftarrow fail + 1$ `// Fails counter`
    end
    `// Update` α, β, δ `and` γ
    if $it = NITER$ **then**
        Update the probability of each α;
        if $|\Theta| \neq 0$ **then**
            $AVSIM \leftarrow \frac{\sum_{\sigma \in \Theta} SIM(\sigma, \Theta)}{|\Theta|}$;
            if $AVSIM < OLDSIM$ **and** $\delta \leq 0.9$ **then**
                $\delta \leftarrow \delta + 0.1$
            else if $\delta \geq 0.1$ **then**
                $\delta \leftarrow \delta - 0.1$
            end
        end
        if $fail < OLDFAIL$ **then**
            $\gamma \leftarrow \gamma - 0.1$; $fit \leftarrow 0$
        else if $\gamma \geq 0.9$ *and* $fit = NINF$ **then**
            exit `// fail`
        else if $\gamma \geq 0.9$ **then**
            $fit \leftarrow fit + 1$
        else
            $\gamma \leftarrow \gamma + 0.1$
        end
        $\beta \leftarrow 1 - \delta - \gamma$; $it \leftarrow 0$; $OLDFAIL \leftarrow fail$; $fail \leftarrow 0$; $OLDSIM \leftarrow AVSIM$
    end
end

5.2. Adaptive greedy evaluation function

Associated with an activity i, the proposed adaptive greedy evaluation function $F(i)$ has three components: priority rule $L(i)$, intensification $I(i)$ and feasibility $G(i)$, each of them with a respective weight

$\beta, \delta, \gamma \in [0, 1]$, such that $\beta+\delta+\gamma = 1$, representing the importance of each component of the evaluation function. The adaptive greedy evaluation function is then defined as follows:

$$F(i) = \beta \times L(i) + \delta \times I(i) + \gamma \times G(i).$$

Priority due to a priority rule $L(i)$: Computational experiments presented in Polo-Mejía et al. (2019a) suggest that using the greedy algorithm with priority rules "Most Successors", "Greatest Rank" and "Longest Duration" provides smaller optimality gaps. Let Γ_i be the set of successors of activity i in precedence constraint set E. We opt for a mixed priority rule that extends the Greatest Rank rule to all successors in Γ_i, i.e.

$$L(i) = p_i + \sum_{j \in \Gamma_i} p_j \ .$$

Intensification component $I(i)$: The idea is to use the characteristics of a set ε of elite solutions to influence the construction phase. In our algorithm, the quality of the solution is highly dependent on the order (σ_s) in which activities are handled by the greedy algorithm to obtain a solution s. Hence ε stores the set of sequences of the elite solutions. Let us define $e(\sigma, i, j)$ as a binary function taking the value of 1 if activity i was treated before activity j in σ, and 0 otherwise. Recall that $\mathcal{Q}$ is be the set of not yet scheduled activities. The intensification component is defined as follows:

$$I(i) = \sum_{\sigma \in \varepsilon} \sum_{j \in \mathcal{Q}} e(\sigma, i, j) \ .$$

The reader should observe that the intensification component favours activities with more successors. For each pair of activities $\{i, j\}$ in each sequence σ_s the value 1 is obtained every time i is scheduled before j. The intensification component sums all these values, thus preferring activities with more successors.

Feasibility factor $G(i)$: As stated at the beginning of the present article, the presence of time windows can make the greedy algorithm fail in obtaining a feasible solution. We propose then to introduce a component giving priority to activities with a short slack time, which allows us to have a greater probability to generate feasible solutions. Let $ES_i \geq r_i$ denote the earliest time activity i can be started taking account of its already scheduled predecessors. Slack time refers to the margin an activity i has in its planning window. The feasibility factor is defined as follows:

$$G(i) = \frac{1}{d_i - ES_i - p_i + 1} \ .$$

Note that $L(i)$, $I(i)$ and $G(i)$ must be normalised before taking the weighted sum, giving $\mathcal{L}(i)$, $\mathcal{I}(i)$ and $\mathcal{G}(i)$, respectively, taking their values in $[0, 1]$. This normalisation is done by dividing the value for an activity by the sum of all activity values. Parameters β, δ and γ are self-adaptive; they are initially equal (to 0.33) and their values are periodically updated. If after $NITER$ GRASP iterations the number of infeasible solutions increases, we must increase the value of γ; on the contrary, if this number decreases, we decrease γ (to try more diverse solutions). If no feasible solution has been found during $NINF$ iterations with the maximum $\gamma \geq 0.9$, the algorithm stops with a failure.

On the other hand, the parameter δ decreases when the diversity of obtained solutions is too low and increases when the variability is high. Finally, β is a function of δ and γ: $\beta = 1 - \delta + \gamma$. To ensure that $\delta + \gamma \leq 1$, we limit δ and γ to be not greater than 1 or lower than 0 (see Algorithm 3 for a better presentation of the self-adaptation scheme).

For measuring the variability (diversity) of the solutions, we keep the sequences used to generate them as reference. Let us define Θ as the set of the last sequences that generated feasible solutions during the last GRASP iterations, and $\Theta^{i \prec j}$ as the set of sequences $\sigma \in \Theta$ where activity i is handled before activity j by the greedy algorithm. $\sigma(v)$ represents the $v-$th element (activity) of sequence σ. For each sequence, we can define a similarity index ($SIM(\sigma, \Theta)$), very close to Kendall's tau distance (Kendall, 1938), indicating the degree on which a sequence σ shares the same characteristics of the other sequences belonging to Θ, as follows:

$$SIM(\sigma, \Theta) = \sum_{v=1}^{|\sigma|-1} \sum_{u=v}^{|\sigma|} |\Theta^{\sigma(v) \prec \sigma(u)}| \ .$$

A reduction on the average value of $SIM(\sigma, \Theta)$ after $NITER$ iterations indicates a better diversity of the solutions.

5.3. Updating the elite solutions set

A warm-up phase is necessary to be able to build the initial set of elite solutions (i.e. the sequences that generate these solutions). During this warm-up phase, all sequences, regardless of the quality of the solutions they generate, will be included in the elite solution set until reaching the number of elite solutions; the only condition to include sequences in ε is that all sequences must be different. Note that during this warm-up phase $F(i) = \beta\mathcal{L}(i) + \gamma\mathcal{G}(i)$ (with $\beta + \gamma = 1$). Parameters β and γ will be self-adapting according to the solution quality and the number of infeasible solutions, respectively.

Once the warm-up phase is finished, the set of elite solutions is updated according to the quality of solutions and their similarity indicator. If a new sequence generates a solution with a $C_{\max}$ lower than the worst $C_{\max}$ in the elite solutions, this new sequence is included in ε. The sequence with the worst $C_{\max}$ is then deleted from the elite set. If several sequences have the worst $C_{\max}$, we delete the sequence with the highest $SIM(\sigma, \varepsilon)$ value to improve the diversity within the elite solution. If the new solution generates a solution with a $C_{\max}$ equal to the worst $C_{\max}$ in ε, we will include the new sequence only if it has a lower $SIM(\sigma, \varepsilon)$ value than the sequences with the worst $C_{\max}$ in the elite set (the sequence with the highest similarity indicator value is deleted).

6. Large neighbourhood search

In this section, we present a Large Neighbourhood Search (LNS) algorithm with an exact solution approach for subproblems inspired by the algorithms proposed by Palpant et al. (2004) for the RCPSP. At

each iteration, starting from a feasible initial solution, the method fixes a part of the current solution, while the other part is solved using an exact method.

In our algorithm, we start with an initial solution obtained by the multi-pass greedy algorithm (Section 3). For generating the subproblem to be solved, we define a sliding *time window* with a fixed length. The length of the window is a function of the average duration of the activities (various length are tested for the computational experiments). At each iteration of the algorithm, all the activities within the time window are selected according to their preemption type to be rescheduled, and all resources and technicians used by them are released and can be redistributed using an exact technique. For our numerical experiments we test and compare both Constraint Programming (CP) and Mixed-Integer Linear Programming (MILP) models presented in Polo-Mejía et al. (2020) for the MSPSP-PP, the latter formulation is recalled in Section 2. For ensuring that the subproblem is solved in reasonable time, we limit the maximal solution time allowed to the solver. The time window is then shifted to the right at each iteration of the algorithm. The first time window starts at $t = 0$, and windows to follow start in the middle of the previous one.

To select the activities that are included in the subproblem, we must look at their preemption type:

- For *non-preemptive* and *partially preemptive* activities, we include all activities for which the interval [start, end] of the activity overlaps the time window. For these type of activities, we select the entire activity to be rescheduled;
- For *preemptive* activities, we only include the time units of the activity that are executed within the time window.

See Appendix E for an illustration of time window activity selection.

The scheduling horizon of the subproblem spans from the earliest start time to the latest end time of selected activities. These start and end times must take into account the precedence relationships with the activities that remain fixed. The fixed activities will also be reflected in the resources and technicians availability. A small modification must be done for the objective function of the models proposed by Polo-Mejía et al. (2020). Instead of always minimising the $C_{\max}$, we must try now to minimise the average end time of activities, except when time window reaches the $C_{\max}$ of the current solution, such as time window TW_3 in Figure E5-c. In the latter case, the objective function is still the minimisation of $C_{\max}$. The new objective function associated to the activities included in TW_q is:

$$\min \sum_{i \in TW_q} C_i \ .$$

If after solving the subproblem we obtain a solution that can lead to improvement (at least the finish time of one of the activities within the time window has decreased), we keep the subproblem solution and insert it to the global problem solution. Using the greedy algorithm, we try to improve the scheduling of activities to the right of the time windows. More precisely, the multi-pass greedy Algorithm 1 is called on these activities, which has the effect to possibly shifting some of these activities to the left thanks to the time slots left free by the activities of the subproblem. Making the schedule more compact may lead to a makespan reduction.

Once these operations are performed, the time window is shifted to the right, and the process is repeated. The heuristic stops when the time window reaches the $C_{\max}$ of the current solution.

As a modification, a multi-sweep version of the algorithm can be carried out. After reaching the $C_{\max}$ of the current solution, the sliding time window is reset to the period t where the first change between the initial solution and the new one occurs. The time window starts to slide again until it reaches the $C_{\max}$. The multi-sweep version stops after two iterations without improving the $C_{\max}$. Algorithm 4 summarises the proposed method.

Algorithm 4: LNS for the MSPSP-PP

```
Generate initial solution using the multi-pass greedy SGS (Algorithm 1);
if a feasible solution has been found then
    Improvement ← True;
    Define initial time window;
    while Improvement do
        Select activities for the subproblem;
        Construct subproblem;
        Solve subproblem using the CP or MILP exact method (Polo-Mejía et al., 2020);
        if Subproblem solution is improved then
            Include subproblem solution in the global solution;
            Reschedule the activities to the right of the current time window with multi-pass
              greedy Algorithm 1
        end
        if C_max is inside the current time window then
            if Current C_max is equal than previous one then
                Improvement ← False;
            else
                Return the time window to period t where first change happened (compared to
                  previous solution)
            end
        else
            Shift time window to start at the middle of the previous one;
        end
    end
end
```

7. Computational experiments

For testing the performance of the proposed heuristic methods, we generate four sets of instances (A, B, C, D) using a basic instance generation procedure that reflects some of the characteristics of a real nuclear research facility. The procedure allows fixing aspects such as portions of preemption type, percentage of activities with time windows, density of precedence relationships, and skill number per technician. Since the portion of preemption type within an instance seems to have a great impact on problem complexity, we analyse the behaviour of the proposed algorithms when this portion changes (see Table 1). As indi-

cated before, the objective of this work is to use heuristic methods to solve problems of real industrial size. Each of the four sets has a total of 50 instances, each of them with 50 activities to be scheduled (average number of activities per week executed in the nuclear facility under study), and an expected $C_{\max}$ ranging from 130 to 170 time units (representing the total number of working hours in a week for the nuclear laboratory). The duration of the activities is from 5 up to 15 time units; they may require up to 15 skills and up to 8 cumulative resources (both following a discrete uniform probability distribution). 20% of the activities are subject to time windows. Each activity may have at most 3 successors. A total of 8 technicians (multi-skilled resources) are available in 2 teams (out of 4 each) doing work-shifts of 12 hours. The skills mastered for each technician are also randomly generated, going from 5 up to 10 skills for each of them. For ensuring the feasibility of an instance, we use a trial and error approach, generating first the instance with the random generator and proving then its feasibility (or not) using the CP Optimizer solver and the CP model presented in Polo-Mejía et al. (2020).

Table 1
Distribution of preemption types for instances of the MSPSP-PP

Preemption type	Set A	Set B	Set C	Set D
Non-preemptive	10%	10%	80%	33.3%
Partially preemptive	10%	80%	10%	33.3%
Preemptive	80%	10%	10%	33.3%

The proposed heuristics are coded in the C++ programming language. To solve the flow problems, we use an adapted procedure of the Edmonds-Karp algorithm proposed by Ababei (2009), which is coded in the C++ programming language. To obtain a lower bound or, in some cases, optimal solutions, we use the MILP and CP models proposed in Polo-Mejía et al. (2020) which are solved using CPLEX 12.7.1 and CP Optimizer 12.7.1 and limiting the solving time to 2 hours (using up to 16 threads and 64 GB of RAM). All computational tests for the heuristics have been carried out using a machine under Ubuntu 16.04.6 operating system, equipped with an Intel Xeon E5-2695 processor running at 2.3 GHz, and RAM limited to 16 GB.

7.1. Tree-based local search

We propose a multi-pass version using the priority rules presented in Section 3.3. We may argue that the computational time for each priority list grows exponentially according the percentage of preemptive activities. However, we control the exponential time complexity with the priority parameters. To get faster results, we propose first to determinate the $C_{\max}$ for every priority list using the multi-pass greedy algorithm. Then, the local search algorithm is executed starting from the list with the smallest $C_{\max}$ to the one having the largest one, keeping always the best $C_{\max}$ as upper bound for cutting branches.

We test the self-adaptive multi-pass version of the tree-based local search algorithm (Algorithm 2), setting the values of π^- and π^+ to 10% and 80% respectively. We do not set any limit on the maximum number of discrepancies ($NBMAXDISCR = +\infty$). Note that we do not set any other stopping criteria neither, since the fact of having a self-adaptive π allows us to perform the search in a reduced time regardless of the instance characteristic. Table 2 shows the average gap values (percentage difference be-

tween the obtained solution and the best known lower bound) for the multi-pass tree-based local search algorithm. It also shows the average gap for the CP model after 5 minutes of computation using CP Optimizer with only one thread. Results obtained with the MILP model are not shown since the solver runs out of memory before giving any feasible solution. We observe that the self-adaptive configuration, presented in this article, allows obtaining less variable average execution times for all sets of instances, compared to those obtained in Polo-Mejía et al. (2019a). The tree-based local search algorithm outperforms the results obtained by the CP solver after 5 minutes of computing time for sets A, B and D, being faster and giving lower average gap. However, for highly non-preemptive instances, set C, the CP solver gives better results. This can be explained by the fact that, for highly non-preemptive instances, the tree-based local search algorithm will explore less feasible solutions.

Table 2
Results for local search algorithm with self-adaptive π

	Local search algorithm		**CP after 5 min**
	Average gap	Average execution time	Average gap
Set A	**4.03%**	89.89 s	6.01%
Set B	**4.78%**	160.35 s	6.65%
Set C	8.77%	115.10 s	**7.65%**
Set D	**4.30%**	193.88 s	5.56%
All	**5.46%**	139.80 s	6.47%

7.2. GRASP

The GRASP algorithm is tested on the set of 200 instances. After testing different configurations for the size of the elite solution set and the maximum number of feasible GRASP iterations, we fix these values to 20 and 550, respectively. This decision is motivated by the objective of keeping the execution time per instance below 90 seconds. In fact, for a fixed number of iterations, a bigger set of elite solutions will increase the execution time, but will not improve significantly the quality of solutions. A smaller set of elite solutions, on the other hand, will negatively impact the quality. To analyse the impact of the intensification component, we test a basic GRASP version where this component is not taken into account, and perform a maximum of 550 feasible GRASP iterations. For the local search phase, we use, in both configurations, the self-adaptive version presented in Section 4 (Algorithm 2), fixing π^- and π^+ to 5% and 60%, respectively. The number of discrepancies by branch is limited to $NBMAXDISCR = 1$. Results are shown in Table 3.

We observe that the GRASP with intensification component (complete GRASP) gives a lower average gap for all sets of instances. The *p-values* for the Wilcoxon Signed Rank Test (Scheff, 2016) show that there is enough evidence to say that the intensification component improved the results obtained for sets B (*p-value*=0.026) and D (*p-value*=0.017). For sets A (*p-value*=0.246) and C (*p-value*=0.1129), the GRASP with intensification is in the worst case equal to its version without intensification. We can then conclude that the intensification component helps to improve the quality of final solutions. Both configurations outperform the CP results for sets A, B and D. CP remains better when the portion of

Table 3
Results for the GRASP algorithm

	Complete GRASP		**Basic GRASP**		**CP after 5 min**
	Average gap	Average execution time	Average gap	Average execution time	Average gap
Set A	**2.21%**	77.76 s	2.32%	73.31 s	6.01%
Set B	**2.51%**	88.75 s	2.77%	91.34 s	6.65%
Set C	8.26%	37.53 s	8.63%	35.94 s	**7.65%**
Set D	**2.98%**	71.22 s	3.30%	68.02 s	5.56%
All	**3.99%**	68.81 s	4.26%	67.15 s	6.47%

non-preemptive activities is high (set C).

7.3. LNS

We test our algorithm using different fixed lengths for the sliding time window, all of them depending of the average duration ($\overline{p}$) of activities within the instances: $0.5\overline{p}$, $\overline{p}$, and $1.5\overline{p}$. The first issue we want to study is the benefit of using CP or MILP to solve the optimisation subproblem; we then tested the single-sweep version of the heuristic using both techniques. Note that we limit the number of threads used by CPLEX and CP Optimizer to 1, and the maximum computing time expended on each optimisation subproblem is 30 sec. The maximum computation time per instance allowed to the entire heuristic is 5 min. We use as an initial solution the best solution obtained by the multi-pass version of the greedy algorithm (Algorithm 1). Results are presented in Table 4.

Table 4
Results for single-sweep LNS using MILP and CP

	MILP						**CP**					
	$0.5\overline{p}$		$\overline{p}$		$1.5\overline{p}$		$0.5\overline{p}$		$\overline{p}$		$1.5\overline{p}$	
	Average gap	Average time	Average gap	Average time	Average gap	Average time	Average gap	Average time	Average gap	Average time	Average gap	Average time
Set A	2.34%	23.36 s	2.18%	49.69 s	**1.97%**	164.9 s	3.47%	234.08 s	3.39%	311.09 s	3.45%	310.46 s
Set B	**2.62%**	231.77 s	3.05%	286.28 s	3.56%	312.54 s	4.69%	310.95 s	4.55%	308.33 s	4.60%	305.13 s
Set C	7.93%	56.95 s	7.85%	80.92 s	**7.53%**	173.17 s	8.20%	88.42 s	8.17%	80.48 s	7.95%	109.59 s
Set D	2.96%	102.12 s	**2.87%**	179.40 s	3.28%	281.73 s	4.32%	311.03 s	3.95%	312.93 s	3.94%	311.65 s
All	**3.97%**	103.56 s	3.99 %	149.07 s	4.09%	233.14 s	5.18%	236.12 s	5.02%	253.21 s	4.99%	259.21 s

We observe that the heuristic using MILP always gives a smaller average gap. It is also most of the time faster than the heuristic using CP. The analysis of the results of all instances does not allow to determine whether the length of the sliding time window has an impact on the quality of the final solution or not. However, if we analyse the results for each set of instances, we observe that for sets A and C (i.e. instances with a low portion of partially preemptive activities) a larger length of the sliding time window reduces the final average gap for the MILP heuristic. On the other hand, if the portion of partially preemptive activities is high (set B), the MILP heuristic gives better results when the length of the time window is smaller. Larger time windows are expected to allow lower gaps, specifically when dealing with a small number of preemptive activities. The more the number of non-preemptive

activities, the higher the need of having wider time windows that contemplate entire activities and that allows the MILP to reallocate resources more efficiently.

Table 5 presents the results for the multi-sweep version of the heuristic. We use MILP to solve the subproblem, as it gives the best and fastest results, with an time limit for subproblem solution of 30 seconds. We test the algorithm using three time windows selection sizes: $0.5\overline{p}$, $\overline{p}$, and $1.5\overline{p}$. We stop algorithm's execution after two complete sweep without improvement. In any case, the maximal time allowed to the whole algorithm is 5 minutes. As expected, there is a small reduction of the average gap for all sets of instances and all time windows length. For set A, we observe a similar behaviour to the one observed during the single-sweep version, i.e. larger time windows lead to lower gaps. Instances from sets B and D have lower average gaps when time windows are small. When compared against the single-sweep version, the execution time is increased by 1.5 times on average, for a reduction of 12% for the average gap.

Table 5
Results for multi-sweep LNS using MILP

	Multi-sweep LNS using MILP					
	$0.5\overline{p}$		$\overline{p}$		$1.5\overline{p}$	
	Average gap	Average time	Average gap	Average time	Average gap	Average time
Set A	2.13%	47.59 s	1.83%	151.43 s	**1.78%**	296.94 s
Set B	**2.44%**	287.51 s	3.01%	312.89 s	3.56%	312.93 s
Set C	7.34%	99.26 s	7.67%	159.04 s	**7.16 %**	279.21 s
Set D	**2.39%**	191.59 s	2.64%	270.46 s	3.24%	310.87 s
All	**3.58%**	156.65 s	3.79%	223.46 s	3.94%	299.99 s

7.4. Combining GRASP and LNS

To improve the obtained results, we combine the GRASP algorithm with a single LNS iteration. We execute first the complete version of the GRASP algorithm using the configuration described before. The LNS algorithm (applying its MILP version) then improves the obtained solution on a single-sweep (we decided to use the single-sweep since it gives a better ratio gap/execution time that the multi-sweep version). We use a fixed time window length of $0.5\overline{p}$ (since it gives the fastest results). Results are presented in Table 6. We observe that combining GRASP and LNS significantly improves the quality of the obtained solutions. This combination outperforms all the solutions obtained by the CP solver within a limited time. Similar to the other proposed heuristics, this algorithm performs better when the portion of non-preemptive activities is low.

7.5. Comparing all methods

Table 7 show a synthesis of the average execution times and average gaps for the different heuristic methods proposed in this article. One observes that the execution time of the multi-pass greedy algorithm

Table 6
Results for LNS after GRASP

	GRASP + LNS (single-sweep)		**CP after 5 min**
	Average gap	Average execution time	Average gap
Set A	**1.56%**	87.29 s	6.01%
Set B	**1.79%**	279.56 s	6.65%
Set C	**6.68%**	82.65 s	7.65%
Set D	**2.06%**	147.97 s	5.56%
All	**3.02%**	149.37 s	6.47%

Table 7
Results for all proposed algorithms

	Gap					**Time (sec)**				
	Set A	Set B	Set C	Set D	All	Set A	Set B	Set C	Set D	All
Multi-pass greedy algorithm	5.51%	6.14%	12.79%	6.51%	7.74%	0.2	0.3	0.4	0.3	0.3
Tree-based local search	4.03%	4.78%	8.77%	4.30%	5.46%	89.9	160.4	115.1	193.88	139.8
Complete GRASP	2.21%	2.51%	8.26%	2.98%	3.99%	77.8	88.8	37.5	71.2	68.8
LNS (MILP) Single-sweep ($0.5\overline{p}$)	2.34%	2.62%	7.93%	2.96%	3.97%	23.4	231.8	56.9	102.1	103.6
LNS (MILP) Multi-sweep ($0.5\overline{p}$)	2.13%	2.44%	7.34%	2.39%	3.58%	47.6	287.5	99.3	191.6	156.7
GRASP + LNS	**1.56%**	**1.79%**	**6.68%**	**2.06%**	**3.02%**	87.3	279.6	82.7	147.9	149.4
CP (1 thread)	6.01%	6.65%	7.65%	5.56%	6.47%	300	300	300	300	300

is significantly lower than all other approaches. However, its average gap is the highest (7.74%) — a typical behaviour given the simplicity of the method. The GRASP method gives the best compromise between execution time and average gap. This method has the second smallest average execution time (68.81 sec) and an average gap of 3.99%. When compared to the approach giving the smallest gap (GRASP + LNS single-sweep), the GRASP is twice faster; and its average gap is only 30% away from the best average gap value. The tree-based local search, on the other hand, gives the worst trade-off between time and quality. This algorithm is slower and results in a worse average gap than the GRASP and LNS single-sweep methods. Even if the local search algorithm is 1.1 times faster than the GRASP+LNS, its average gap is 79% worse than the average gap of the GRASP+LNS algorithm. However, it largely improves the results of the greedy algorithm and it is also a key component of the efficient GRASP approach.

7.6. Comparing iterative methods within a time budget

The GRASP, LNS, and the hybrid GRASP+LNS procedures are iterative methods that can be stopped with a time limit criterion. To fairly compare these methods, we perform them with a global 300 sec time limit. The results, in terms of average gap from the best known solution on each instance family, and the number of times each method is able to reach the best known solution on each instance family, are displayed in Table 8. For the hybrid GRASP+LNS approach, we test different combinations to meet the global 5 min time budget. The results show that LNS performs better than the GRASP under this budget. The best combination is the hybrid approach using 2 min for the GRASP and 3 min for LNS or 1 min for

the GRASP and 4 min for LNS. The improvement upon the CP method is much larger for all instance families, including the strongly non-preemptive ones, when the methods are given the same time budget.

Table 8
Results for the proposed iterative algorithms within a 300 sec time budget

	Gap					**Number of best solutions found**				
	Set A	Set B	Set C	Set D	All	Set A	Set B	Set C	Set D	All
Complete GRASP	2.07%	2.21%	7.43%	2.86%	3.65%	5	20	11	9	45
LNS (MILP) Multi-sweep ($0.5\overline{p}$)	1.25%	2.24%	6.79%	2.18%	3.13%	20	13	17	23	73
GRASP (240s) + LNS(60s)	1.35%	1.97%	6.16%	2.17%	2.92%	17	20	28	22	87
GRASP (180s) + LNS(120s)	1.14%	1.95%	5.89%	1.99%	2.75%	25	21	34	26	106
GRASP (120s) + LNS(180s)	0.91%	**1.66%**	**5.83%**	**1.90%**	**2.58%**	33	**28**	**35**	29	125
GRASP (60s) + LNS(240s)	**0.83%**	1.90%	**5.83%**	1.92%	2.63%	**37**	25	**35**	**31**	**128**

7.7. Experimental results on standard MSPSP instances from the literature

Since the GRASP algorithm provides the desired compromise between execution time and average gap, we test its performance on conventional MSPSP instances. To that purpose, we need to make a few changes to the greedy algorithm (Algorithm 1) which is used by the GRASP and also by the tree-based local search method (Algorithm 2) used inside GRASP. The MCMF problem has to be changed in two ways: first there is no need to include the $L + 1$ skill node as there is not a minimum number of required operators. Second, one operator must be allocated to at most one skill. This is simply achieved in the MCMF model by setting the maximal capacity of the arc issued from the operator to the sink to 1. Also, in the search for feasible intervals, the condition for operator allocation feasibility at line 17 of Algorithm 1 is not sufficient anymore and the feasibility of the MCMF problem must be checked instead.

We use as reference the instances and results presented by Young et al. (2017). The instances are decomposed into four sets. The instances of Set 1A, generated with the same parameters as the instances proposed by Correia et al. (2012), have 22 activities, 4 skills and from 10 to 30 resources. Set 1B is generated using the parameters of the instance set originally proposed by Almeida et al. (2016) and it contains instances with 42 activities, 4 skills and from 20 to 60 resources. Sets 2A, 2B and 2C are taken from Montoya et al. (2014) and the instances have from 20 to 62 activities, from 2 to 15 skills and from 5 to 19 resources.

Table 9 shows the results obtained by our GRASP algorithm. All the parameters used for solving the literature instances, are the same as presented for the complete GRASP in Section 7.2 Column "Gap for GRASP" provides the average gap from the best known solution (Young et al., 2017) on the different instance sets. For instances of Set 1A and Set 1B, GRASP is able to find solutions with an average gap smaller than 3%. Regarding the instances of Set 1B, the GRASP algorithm is 3.5 times faster than the algorithm proposed by Young et al. (2017)[2]. In addition, the GRASP algorithm is also able to improve 56 of the best solutions ever found within this set. For instances in Sets 2A, 2B and 2C, our algorithm

[2]Their experiments were run on a PC with an Intel i7 2600 CPU 3.4 GHz and 8 GB of memory, which represents a computer architecture comparable with ours.

give solutions[3] with average gap lower than 5.3%.

This demonstrates the quality of the proposed algorithm, even for the particular case of the considered problem. The use of a memory intensification component, mostly missing in the heuristics proposed for the MSPSP, is the key factor for the success of our GRASP algorithm on these standard instances. The inclusion of this component improves the quality of solutions, as demonstrated in Section 7.2.

Table 9
Results for the MSPSP instances of Young et al. (2017)

		Execution time (sec)	
	Gap for GRASP	*GRASP*	*Young et al. (2017)*
Set 1A	2.8%	40.3	0.5
Set 1B	2.3%	151.1	536.3
Set 2A	4.7%	67.6	196.6
Set 2B	4.3%	58.9	122.8
Set 2C	5.21%	68.6	1.2

8. Concluding remarks

In this article, we presented various heuristic and metaheuristic methods for solving the multi-skill project scheduling problem with partial preemption (MSPSP-PP). Initially, a greedy algorithm was proposed. At each iteration, the greedy algorithm decomposed the MSPSP-PP into two subproblems: scheduling and resource allocation. For the scheduling part, a priority rule was used. The allocation subproblem was solved using a minimum-cost maximum-flow problem. Computational tests showed that the greedy algorithm allows obtaining solutions (in time lower than 1 second) that are close to the ones obtained by a CP solver after 5 min of computing.

A tree-based local search algorithm, partially inspired by limited discrepancy search, was also proposed to improve the solutions obtained by the greedy algorithm. The self-adaptive version of this local search algorithm allowed obtaining solutions with an average gap of 5.5%.

A greedy randomised adaptive search procedure (GRASP), combining the greedy and local search algorithms, has been then introduced. The proposed GRASP included some components looking for an improvement of the solution feasibility, and the use of characteristics of best found solutions to bias the generation of new solutions (intensification). The GRASP provided good results during the computational test, being able to obtain solutions with an average gap of only 4%. The use of intensification in the GRASP showed an improvement of the quality of solutions when the portion of partially preemptive activities is high. The GRASP was the method giving the best trade-off between execution time and solution quality.

Finally, an LNS-algorithm was also proposed. The LNS scheme hybridises the greedy algorithm with the exact resolution of an optimisation subproblem. Computational tests showed that the use of MILP for solving the optimisation subproblem allowed obtaining better solutions faster (compared with the use of CP for the subproblem exact resolution). The LNS method allowed obtaining solutions with an

[3]These solutions are publicly available at: https://homepages.laas.fr/lopez/PUBLIC/ImprovedYoungSolutions.zip .

average gap of 3.5%. Executing LNS after GRASP provided better results (average gap of 3%), largely outperforming the results of the CP solver (with time limited to 5 min).

To validate our approach we slightly modified the components used in the GRASP heuristic to be able to deal with standard MSPSP instances. On the instances proposed by Young et al. (2017), the GRASP remains under an average gap of 5.21% from the best known solution and is able to find 56 new best solutions in a reasonable time. All proposed methods were negatively affected by a high portion of non-preemptive activities (higher average gaps).

Further research should be carried out to improve the quality of solutions when this portion of non-preemptive is substantial. Special attention must be given to the study of a better local search algorithm since it plays an important role in all the proposed methods. The local search algorithm proposed in this article has an exponential time complexity. In addition, we cannot guarantee that it always finds the local optimum of the neighbour. The parallelisation of our methods is another way of improvement, since it will allow reducing the execution time or increasing the number of generated solutions and the size of the explored neighbours without increasing the current execution time (Alba, 2005). Finally, a worthwhile research direction would be to consider two-phase heuristics where the schedule of all activities satisfying renewable resource requirements is solved first and operator allocation is carried out in the second phase which is the principle of the method proposed by Bellenguez-Morineau and Néron (2007).

References

Ababei, C., 2009. C++ adapted version of the Edmonds-Karp relabelling MCMF algorithm. https://github.com/eigenpi/MCMF4. Online; last accessed September 1, 2018.

Ababei, C., Kavasseri, R., 2010. Efficient network reconfiguration using minimum cost maximum flow-based branch exchanges and random walks-based loss estimations. *IEEE Transactions on Power Systems* 26, 1, 30–37.

Alba, E., 2005. *Parallel metaheuristics: A new class of algorithms*. Wiley-Interscience, USA.

Almeida, B.F., 2018. Multi-skill resource-constrained project scheduling problems: models and algorithms. Ph.D. thesis, Universidade de Lisboa, Faculdade de Ciências, Lisbon.

Almeida, B.F., Correia, I., Saldanha-da Gama, F., 2016. Priority-based heuristics for the multi-skill resource constrained project scheduling problem. *Expert Systems with Applications* 57, 91–103.

Almeida, B.F., Correia, I., Saldanha-da Gama, F., 2018. A biased random-key genetic algorithm for the project scheduling problem with flexible resources. *Top* 26, 2, 283–308.

Almeida, B.F., Correia, I., Saldanha-da Gama, F., 2019. Modeling frameworks for the multi-skill resource-constrained project scheduling problem: a theoretical and empirical comparison. *International Transactions in Operational Research* 26, 3, 946–967.

Artigues, C., 2008. The resource-constrained project scheduling problem. In Artigues, C., Demassey, S. and Néron, E. (eds), *Resource-constrained project scheduling: models, algorithms, extensions and applications*. John Wiley & Sons, pp. 21–36.

Bellenguez-Morineau, O., 2006. Méthodes de résolution pour un problème de gestion de projet multi-compétence. Ph.D. thesis, Université François Rabelais, Tours.

Bellenguez-Morineau, O., 2008. Methods to solve multi-skill project scheduling problem. *4OR* 6, 1, 85–88.

Bellenguez-Morineau, O., Néron, E., 2007. A branch-and-bound method for solving multi-skill project scheduling problem. *RAIRO-Operations Research* 41, 2, 155–170.

Correia, I., Saldanha-da Gama, F., 2015. A modeling framework for project staffing and scheduling problems. In Schwindt, C. and Zimmermann, J. (eds), *Handbook on Project Management and Scheduling - Volume 1*. Springer, pp. 547–564.

Correia, I., Lourenço, L.L., Saldanha-da Gama, F., 2012. Project scheduling with flexible resources: formulation and inequalities. *OR spectrum* 34, 3, 635–663.

Festa, P., Resende, M.G., 2009a. An annotated bibliography of GRASP–Part I: Algorithms. *International Transactions in Operational Research* 16, 1, 1–24.

Festa, P., Resende, M.G., 2009b. An annotated bibliography of GRASP–Part II: Applications. *International Transactions in Operational Research* 16, 2, 131–172.

Goldberg, A.V., 1997. An efficient implementation of a scaling minimum-cost flow algorithm. *Journal of Algorithms* 22, 1, 1–29.

Harvey, W.D., Ginsberg, M.L., 1995. Limited discrepancy search. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95) - Volume 1*, Montreal, Quebec, Canada, pp. 607–613.

Javanmard, S., Afshar-Nadjafi, B., Niaki, S.T.A., 2017. Preemptive multi-skilled resource investment project scheduling problem: Mathematical modelling and solution approaches. *Computers & Chemical Engineering* 96, 55–68.

Kendall, M.G., 1938. A new measure of rank correlation. *Biometrika* 30, 1/2, 81–93.

Kolisch, R., 1996. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research* 90, 2, 320 – 333.

Kolisch, R., Heimerl, C., 2012. An efficient metaheuristic for integrated scheduling and staffing IT projects based on a generalized minimum cost flow network. *Naval Research Logistics* 59, 2, 111–127.

Li, H., Womer, K., 2009. Scheduling projects with multi-skilled personnel by a hybrid MILP/CP Benders decomposition algorithm. *Journal of Scheduling* 12, 3, 281.

Lin, J., Zhu, L., Gao, K., 2020. A genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem. *Expert Systems with Applications* 140, 112915.

Montoya, C., Bellenguez-Morineau, O., Pinson, E., Rivreau, D., 2014. Branch-and-price approach for the multi-skill project scheduling problem. *Optimization Letters* 8, 5, 1721–1734.

Myszkowski, P.B., Olech, Ł.P., Laszczyk, M., Skowroński, M.E., 2018. Hybrid differential evolution and greedy algorithm (DEGR) for solving multi-skill resource-constrained project scheduling problem. *Applied Soft Computing* 62, 1–14.

Myszkowski, P.B., Siemieński, J.J., 2016. GRASP applied to multi-skill resource-constrained project scheduling problem. In *Proceedings of the International Conference on Computational Collective Intelligence (ICCI 2016)*, Springer, Halkidiki, Greece, pp. 402–411.

Myszkowski, P.B., Skowroński, M.E., Podlodowski, Ł., 2013. Novel heuristic solutions for multi-skill resource-constrained project scheduling problem. In *Proceedings of the 2013 Federated Conference on Computer Science and Information Systems*, IEEE, pp. 159–166.

Néron, E., 2002. Lower bounds for the multi-skill project scheduling problem. In *Proceedings of the 18th International Workshop on Project Management and Scheduling (PMS 2002)*, Valencia, Spain, pp. 274–277.

Noori, S., Taghizadeh, K., 2018. Multi-mode resource constrained project scheduling problem: A survey of variants, extensions, and methods. *International Journal of Industrial Engineering & Production Research* 29, 3, 293–320.

Palpant, M., Artigues, C., Michelon, P., 2004. LSSPER: Solving the resource-constrained project scheduling problem with large neighbourhood search. *Annals of Operations Research* 131, 1-4, 237–257.

Panwalkar, S.S., Iskander, W., 1977. A survey of scheduling rules. *Operations Research* 25, 1, 45–61.

Polo-Mejía, O., Anselmet, M.C., Artigues, C., Lopez, P., 2018a. Mixed-integer and constraint programming formulations for a multi-skill project scheduling problem with partial preemption. In *Proceedings of the 12th International Conference on Modelling, Optimization and Simulation (MOSIM 2018)*, Toulouse, France, pp. 367–374.

Polo-Mejía, O., Anselmet, M.C., Artigues, C., Lopez, P., 2018b. Multi-skill project scheduling in a nuclear research facility. In *Proceedings of the 16th International Conference on Project Management and Scheduling (PMS 2018)*, Rome, Italy, pp. 367–374.

Polo-Mejía, O., Artigues, C., Lopez, P., 2019a. A heuristic method for the multi-skill project scheduling problem with partial preemption. In *Proceedings of the 8th International Conference on Operations Research and Enterprise Systems (ICORES 2019)*, Prague, Czech Republic, pp. 111–120.

Polo-Mejía, O., Artigues, C., Lopez, P., Mönch, L., 2019b. Memory and feasibility indicators in GRASP for multi-skill project scheduling with partial preemption. In *13th Metaheuristics International Conference (MIC 2019)*, Cartagena, Colombia, pp. 153–156.

Polo-Mejía, O., Artigues, C., Lopez, P., Basini, V., 2020. Mixed-integer/linear and constraint programming approaches for

activity scheduling in a nuclear research facility. *International Journal of Production Research* 58, 23, 7149–7166.
Prais, M., Ribeiro, C.C., 2000. Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing* 12, 3, 164–176.
Resende, M.G.C., Ribeiro, C.C., 2019. Greedy randomized adaptive search procedures: Advances and extensions. In *Handbook of Metaheuristics*. Springer, pp. 169–220.
Scheff, S.W., 2016. Nonparametric Statistics, Academic Press, chapter 8. pp. 157–182.
Voß, S., Fink, A., Duin, C., 2005. Looking ahead with the pilot method. *Annals of Operations Research* 136, 1, 285–302.
Young, K.D., Feydy, T., Schutt, A., 2017. Constraint programming applied to the Multi-Skill Project Scheduling Problem. In *Principles and Practice of Constraint Programming, 23rd International Conference (CP 2017)*, Springer, Cham, Melbourne, Australia, pp. 308–317.

Appendix A: An MSPSP-PP instance (running example)

Table A1 illustrates an example of an MSPSP-PP instance with four activities (A_1, A_2, A_3, A_4), a single resource (R_1), two operators (O_1, O_2) and four skills (l_1, l_2, l_3, l_4). Note that this example will serve as a running example throughout the paper. All activities require 1 unit of resource R_1. A_1 is preemptive, A_2 and A_4 are non-preemptive and A_3 is partially preemptive in the sense that R_1 cannot be released during its preemption. Processing times, release dates, and deadlines are also displayed, if any. Skill requirements are given so that only non-zero requirements are displayed, i.e. $a_{11} = a_{23} = a_{24} = a_{32} = a_{43} = 1$. We assume that each activity requires at least one operator $q_1 = q_2 = q_3 = q_4 = 1$. The skills mastered by the operators are also given with $m_{11} = m_{13} = m_{21} = m_{22} = m_{24} = 1$. Resource R_1 has capacity $B_{1t} = 2, \forall t \in \mathcal{T}$.

A feasible solution with makespan 7 is shown in Figure A1. At the left part of the figure, an activity-oriented Gantt chart gives the execution periods of each activity. For the partially preemptive activity A_3 the light-yellow interval corresponds to the time period where the activity is preempted while resource R_1 is not released. The resource-oriented Gantt chart of the right part indicates the resource and operators occupation by the activities. At time period 3, the non-preemptive activity A_2 must be started to meet its deadline. The activities A_1 and A_3 must be preempted since the required skill l_3 is only mastered by operator O_1 and l_4 only by operator O_2. The Gantt chart shows that, contrarily to operators O_1 and O_2, resource R_1 is not released during this preemption, which illustrates the concept of partial preemption. At period 6, A_4 can start and operator O_1 is the only one mastering skill l_3, required by the activity. Thus, operator O_1 switches to A_4 while O_2 switches to A_1 as it also masters skill l_1. This illustrates the possibility to change the operators at any time for preemptive or partially preemptive activities.

Another specific feature of our problem is that there is no constraint that prevents an operator from being employed for several skills simultaneously. Assume for instance that operator O_2 masters also skill l_3 in the example from Table A1. Then, since the minimum number of required operators for activity A_2 is $q_2 = 1$, both skills l_3 and l_4 could be covered by operator O_2. There is a decoupling between the number of required operators for each skill and the global minimum number of required operators for a task. This is motivated by the industrial context of our study. However, the standard MSPSP constraint that only a single operator can be used for a single skill at a given time can easily be incorporated in our models and algorithms (see our results on the standard MSPSP in Section 7.7).

Table A1
An instance for the multi-skill project scheduling problem with partial preemption (MSPSP-PP)

Activity	Duration	(Required skill, Quantity)	(Required resource, Quantity)	Deadline	Release date	Type
A_1	5	$(l_1, 1)$	$(R_1, 1)$	–	–	P
A_2	1	$(l_3, 1), (l_4, 1)$	$(R_1, 1)$	3	3	NP
A_3	3	$(l_2, 1)$	$(R_1, 1)$	–	–	PP (R_1 cannot be released: $\rho_{31} = 1$)
A_4	2	$(l_3, 1)$	–	–	6	NP

	Operator	Mastered skills	Resource	Capacity
	O_1	$\{l_1, l_3\}$	R_1	2
	O_2	$\{l_1, l_2, l_4\}$		

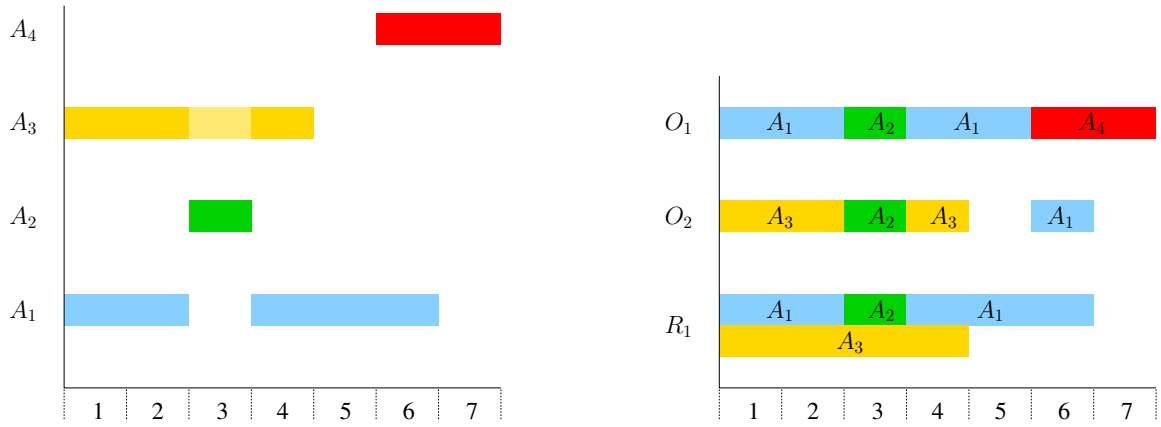

Fig. A1. A feasible solution for the instance from Table A1

Appendix B: Serial SGS illustration

We illustrate an iteration of the greedy algorithm for the Table A1/Figure A1 example. We consider a stage of the greedy algorithm where all activities shown in Figure A1 are scheduled except preemptive activity A_1 and activity A_4. Hence, the timetable $\overline{\mathcal{I}}$ is made of intervals $\overline{I}_1 = [\![1, 2]\!]$ with operator

availability $\bar{A}_{11} = 1$, $\bar{A}_{21} = 0$, $\bar{B}_{11} = 1$, $\overline{I}_2 = [\![3, 3]\!]$ with $\bar{A}_{12} = \bar{A}_{22} = \bar{B}_{12} = 0$, $\overline{I}_3 = [\![4, 4]\!]$ with $\bar{A}_{13} = 1$, $\bar{A}_{23} = 0$, $\bar{B}_{13} = 1$, $\overline{I}_4 = [\![5, T]\!]$ with $\bar{A}_{14} = 1$, $\bar{A}_{24} = 1$, $\bar{B}_{14} = 2$.

Activity A_1 requires skill l_1 which is mastered by both operators O_1 and O_2. Given that A_1 requires also one time unit of resource R_1, the serial SGS would schedule A_1 over intervals $\mathcal{T}_i = \{[\![1, 2]\!], [\![4, 4]\!], [\![5, 6]\!]\}$. Now the MCMF algorithm is applied on each of these intervals to allocate the operators. Figure B2 displays the minimum-cost maximum-flow problem for allocating operators O_1 and O_2 in interval $[\![5, 6]\!]$. Values above the arcs are the arc capacities while the values below the arc are the arc cost. Node l_5 corresponds to the dummy skill for obtaining at least $q_1 = 1$ operator. The operator criticality costs are computed using correlation factors $g_{11} = |\{l_1\} \cap \{l_1, l_3\}| = 1$ and $g_{12} = |\{l_1\} \cap \{l_1, l_2, l_4\}| = 1$. Then the criticality of operator 1, which may be required by the unscheduled activity A_4, is equal to $c_1 = p_4 g_{41}/g_{11} = 2$ while the criticality of operator 2, which is no more required, is $c_2 = 0$. Hence in this situation the serial SGS would allocate operator O_1 to A_1 in interval $[\![5, 6]\!]$. Note that the schedule shown in Figure A1 rather corresponds to order A_2, A_3, A_4, A_1 as operator O_1 is selected in interval $[\![5, 6]\!]$, being in this case of criticality 0.

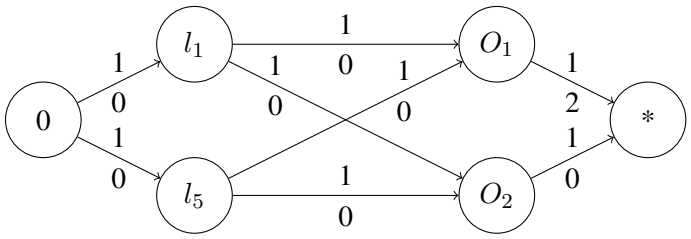

Fig. B2. An MCMF problem for operator allocation

Appendix C: Tree-based local search illustration

C.1. Local search tree

Let us take again the example proposed in Table A1, and suppose we have started our algorithm by scheduling activity A_2 at time $r_2 = 3$ (no alternative is possible for A_2 assignment to the operators). Now the next activity to be scheduled is the preemptive activity A_1. The greedy SGS selects for A_1 execution intervals $\mathcal{T}_i = \{[\![1, 2]\!], [\![4, 6]\!]\}$. Let us compute the operator criticality indicator to determine the preferred allocation for each interval. As already established, correlation indicators for operator 1 are such that $g_{11} = g_{41} = 1$ and $g_{31} = 0$. For operator 2, we have $g_{12} = g_{32} = 1$ and $g_{4,2} = 0$. It follows that the operator criticalities are $c_1 = p_4 = 2$ and $c_2 = p_3 = 3$. Hence operator O_2 is the most critical operator, which means that at this iteration the preferred allocation for A_1 is O_1. As we have two intervals in $\mathcal{T}_1$, this yields 2^2 operator allocations, which can be sorted by preference order: $\{\{O_1\}, \{O_1\}\}, \{\{O_1\}, \{O_2\}\}, \{\{O_2\}, \{O_1\}\}\}, \{\{O_2\}, \{O_2\}\}\}$. The process can be recursively applied to build a search tree enumerating different possible operator allocations and resulting schedules. The method remains a heuristic as the activity selection process follows a priority rule. Figure C3 displays the search tree in case the greedy algorithm selects the first three activities in the order A_2, A_1, A_3. Selected

intervals $\mathcal{T}_i$ are depicted near each branch and the selected allocation for each of them is displayed inside the node. Nodes are numbered in the order of exploration in a depth-first search. We see the impact of operator allocation on the schedule by remarking that intervals selected by activity A_3 depend on the availability of operator O_2. Hence the method can be seen as a tree-based local search based on a priority rule used to select the activities.

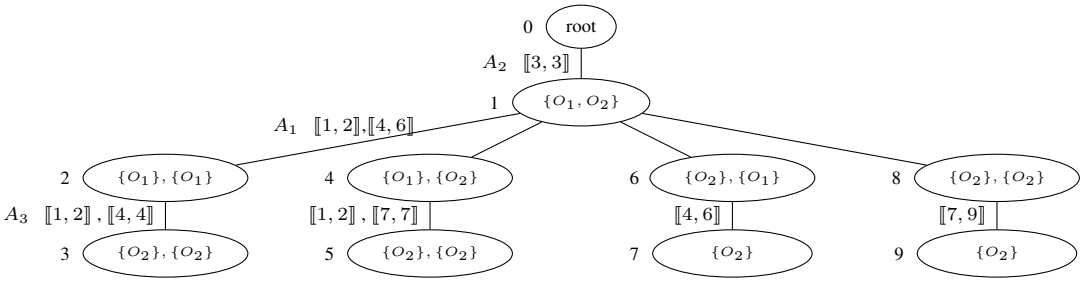

Fig. C3. Local search tree

C.2. An equivalent binary tree

The binary search tree equivalent to the tree of Figure C3 is displayed in Figure C4. We see that the four alternative operator allocations (nodes 2, 4, 6, 8 in Figure C3) for activity A_1 that is preemptively scheduled in two execution intervals $[\![1,2]\!]$ and $[\![4,6]\!]$ can be enumerated by a binary tree of 6 nodes and 4 leaves starting for the 2 possible allocations for the first execution interval $A_{1,1}$ and then enumerating for each of them the 2 possible allocations for the second execution interval $A_{1,2}$. Suppose that the maximal depth of binary tree is $\Delta^{\max} = 4$, the probability of node 4 is $\pi(1 - 3/\Delta^{\max}) = \pi/4$ as the depth of the alternative allocation $\{O_2\}$ for interval $[\![4,6]\!]$ of A_1 in the binary tree is 3. The probability of node 6 is $\pi(1 - 2/\Delta^{\max}) = \pi/2$, which is lower as the alternative $\{O_2\}$ appears deeper in the binary tree. The probability of node 8 is the product of the latter two, i.e. $\pi^2/8$.

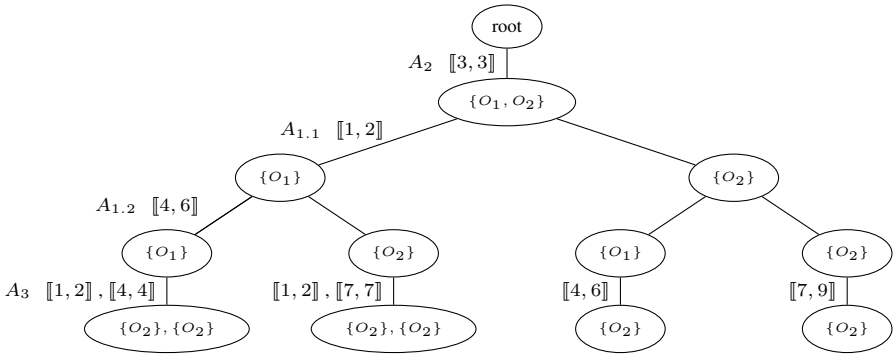

Fig. C4. Binary local search tree equivalent to the non-binary tree of Figure C3

Appendix D: Illustration of an iteration of Algorithm 2

Let us illustrate an iteration of Algorithm 2 for the Table A1/Figure A1 example. We assume that the node extracted from $\mathcal{N}$ at line 5 corresponds to the partial schedule including only activity A_2. The information carried by the node is the following: unscheduled activities $\mathcal{Q} = \{A_1, A_3, A_4\}$, partial schedule $\mathcal{T}_2 = \{[\![3,3]\!]\}$, operator allocations $\mathcal{O}_2 = \{O_1, O_2\}$, timetable $\overline{\mathcal{I}}$ with 3 intervals: interval $\overline{I}_1 = [\![1,2]\!]$ with operator availability $\bar{A}_{11} = \bar{A}_{21} = 1$, $\bar{B}_{11} = 2$, $\overline{I}_2 = [\![3,3]\!]$ with $\bar{A}_{12} = \bar{A}_{22} = \bar{B}_{12} = 0$, $\overline{I}_3 = [\![4,T]\!]$ with $\bar{A}_{13} = A_{23} = 1$, $\bar{B}_{13} = 2$. Moreover, the depth is $\Delta = 1$ and the number of discrepancies is $\#discr = 0$. Suppose that activity A_1 is selected from being scheduled at step 6. As already established, there are two execution intervals computed by Algorithm 1 at line 7, which are $\mathcal{T}_1$ $\{[\![1,2]\!], [\![4,6]\!]\}$ and the preferred and alternative operator allocation are the same for both intervals $\mathcal{O}_{11} = \mathcal{O}_{12} = \{O_1\}$ and $\mathcal{O}'_{11} = \mathcal{O}'_{12} = \{O_1\}$. We illustrate how the binary search tree will enumerate the allocations. The initial node in $\mathcal{B}$ popped at line 16 is such that $r = 0$ and $\#ldiscr = 0$. Therefore the interval index r is set to 1 at line 22, and we assume that the random number obtained at line 23 exceeds the probability $2\pi/\Delta^{\max}$. Consequently, the allocation $\overline{\mathcal{O}}_{11}$ for interval 1 is set to the alternative allocation O_2 and the first right child node ($r = 1, \overline{\mathcal{O}}_{11} = \{O_2\}, \#ldiscr + 1 = 1$) is pushed to $\mathcal{B}$ (line 25), assuming $NBMAXDISCR > 1$. The allocation $\overline{\mathcal{O}}_{11}$ for interval 1 is set to the preferred one $\{O_1\}$ and the first left child node ($r = 1, \overline{\mathcal{O}}_{11} = \{O_1\}, \#ldiscr = 0$) is pushed to $\mathcal{B}$ (line 28). At the next iteration, the same node is popped from $\mathcal{B}$ and r is set to 2 at line 22. We still assume that the random number obtained at line 23 exceeds the probability $3\pi/\Delta^{\max}$, hence the allocation $\overline{\mathcal{O}}_{12}$ for interval $r = 2$ is set to the alternative allocation O_2 and the second right child node ($r = 2, \overline{\mathcal{O}}_{11} = \{O_1\}, \overline{\mathcal{O}}_{12} = \{O_2\}, \#ldiscr + 1 = 1$) is pushed to $\mathcal{B}$. The allocation $\overline{\mathcal{O}}_{12}$ for interval $r = 2$ is then set to the preferred one $\{O_1\}$ and the second left child node ($r = 2, \overline{\mathcal{O}}_{11} = \{O_1\}, \overline{\mathcal{O}}_{12} = \{O_1\}, \#ldiscr = 0$) is pushed to $\mathcal{B}$. At the next iteration, this node is popped from $\mathcal{B}$ and since $r = 2$ we have a leaf node of the binary tree. We consequently push a new node in the temporary stack $\mathcal{N}^{temp}$ with updated information ($\mathcal{Q} = \{A_3, A_3\}$, $\mathcal{T}_2 = \{[\![3,3]\!]\}$, $\mathcal{O}_2 = \{O_1, O_2\}$, $\mathcal{T}_1 = \{[\![1,2]\!], [\![4,6]\!]\}$, $\mathcal{O}_{11} = \{O_1\}$, $\mathcal{O}_{12} = \{O_1\}$, $\overline{I}_1 = [\![1,2]\!]$, $\overline{I}_2 = [\![3,3]\!]$, $\overline{I}_3 = [\![4,6]\!]$, $\overline{I}_4 = [\![7,T]\!]$, $\bar{A}_{11} = 0$, $\bar{A}_{21} = 1$, $\bar{B}_{11} = 1$, $\bar{A}_{12} = \bar{A}_{22} = \bar{B}_{12} = 0$, $\bar{A}_{13} = 0$, $\bar{A}_{23} = 1$, $\bar{B}_{13} = 1$, $\bar{A}_{14} = A_{24} = 1$, $\bar{B}_{14} = 2$, $\delta = 3$, $\#discr = 0$). This corresponds to the node child where all preferred operator allocations are selected for the intervals of activity A_1. The next popped node from $\mathcal{B}$ is also a leaf node of the binary tree ($r = 2, \overline{\mathcal{O}}_{11} = \{O_1\}, \overline{\mathcal{O}}_{12} = \{O_2\}$). Thus, another node is pushed to $\mathcal{N}^{temp}$, corresponding to the preferred allocation for the first interval and the alternative allocation for the second interval. The process ends when all 2^2 nodes are inserted in $\mathcal{N}^{temp}$ (or discarded due to the probabilities). Then, all nodes of $\mathcal{N}^{temp}$ are pushed to $\mathcal{N}$ in reverse order to ensure that the node with all preferred allocations shall be popped first.

Appendix E: Time window activity selection for large neighborhood search

Let us take the solution proposed in Figure A1 for the MSPSP-PP example of Table A1. Let suppose we use a time selection window with size 4 time units. The first time window (TW_1) will go from 1 to 4 (Figure E5-a), activities A_2 and A_3 are completely within TW_1, they will thus be included in the optimisation subproblem. Activity A_1 is preemptive, we choose then only the 3 time units within TW_1. Activity A_4 is outside TW_1 and thus not included in the optimisation subproblem. Once solved the

optimisation subproblem (let suppose for our example no amelioration was found), we must shift the time window to the right, and put its start at the middle of the previous one. In our example, the second time window (TW_2) will go then from 3 to 6 (see Figure E5-b). Activities A_3 and A_4 are partially included within TW_2, however, since they are non-preemptive and partially-preemptive respectively, the whole activities must be included in the optimisation subproblem. Activity A_2 is also totally included, while for activity A_1 only the last three time units are included in the optimisation subproblem. Once again, after solving the optimisation subproblem, shift to the right the time windows (Figure E5-c) and repeat the selection and solution actions.

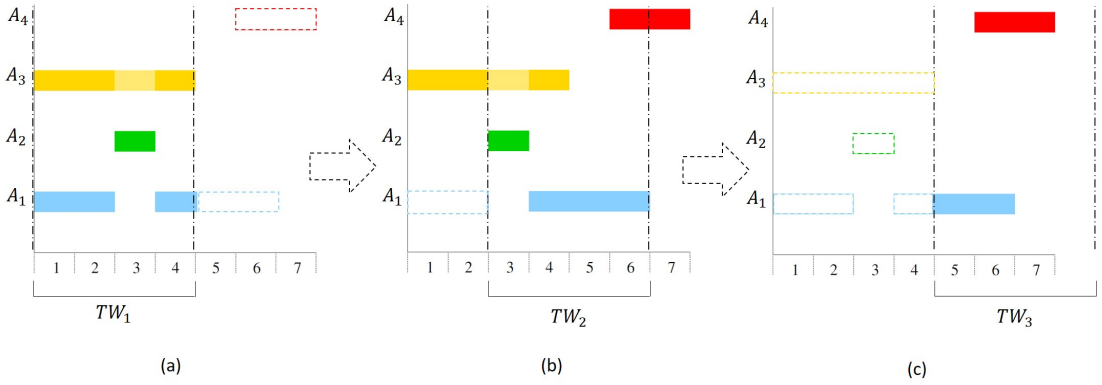

Fig. E5. Time window activity selection