Partially Preemptive Multi Skill/Mode Resource-Constrained Project Scheduling with Generalized Precedence Relations and Calendars

Guillaume Povéda
Airbus (AI Research), Toulouse, France

Nahum Alvarez
Airbus (AI Research), Toulouse, France

Christian Artigues
LAAS-CNRS, Universite de Toulouse, CNRS, Toulouse, France

Abstract

Multi skill resource-constrained project scheduling Problems (MS-RCPSP) have been object of studies from many years. Also, preemption is an important feature of real-life scheduling models. However, very little research has been investigated concerning MS-RCPSPs including preemption, and even less research moving out from academic benchmarks to real problem solving. In this paper we present a solution to those problems based on a hybrid method derived from large neighborhood search incorporating constraint programming components tailored to deal with complex scheduling constraints. We also present a constraint programming model adapted to preemption. The methods are implemented in a new open source python library allowing to easily reuse existing modeling languages and solvers. We evaluate the methods on an industrial case study from aircraft manufacturing including additional complicating constraints such as generalized precedence relations, resource calendars and partial preemption on which the standard CP Optimizer solver, even with the preemption-specific model, is unable to provide solutions in reasonable times. The large neighborhood search method is also able to find new best solutions on standard multi-skill project scheduling instances, performing better than a reference method from the literature.

2012 ACM Subject Classification Computing methodologies → Planning and scheduling; Applied computing → Industry and manufacturing; Theory of computation → Optimization with randomized search heuristics; Theory of computation → Constraint and logic programming

Keywords and phrases Large-scale scheduling problem, partial preemption, multi-skill, multi-mode, resource calendars, constraint programming, large neighborhood search

Digital Object Identifier 10.4230/LIPIcs.CP.2023.31

Funding This work received the support of the Support from the French ANR-3IA Artificial and Natural Intelligence Toulouse Institute (ANITI).
Guillaume Povéda: ANITI
Nahum Alvarez: ANITI
Christian Artigues: ANITI

Acknowledgements The authors are grateful to the anonymous reviewers for their constructive comments. In particular, we thank the anonymous reviewer that pointed out the issue linked to preemption in the *CP-Base* model and made inspiring suggestions that leaded to the *CP-SmartPreemption* model.

1 Introduction

Multi skill resource-constrained project scheduling problems (from hereon MS-RCPSP) are critical in business applications like automotive, human resources, aerospace or nuclear industry [3]; managing efficiently the workforce assigned to perform required tasks directly

impacts the performance and progress of such businesses. This type of problems have been widely studied in academia from long, as well as the more general yet less compact multi-mode model involving non renewable resources [36]. Also, preemption in scheduling is found in many works due to the flexibility it brings to improve the performance of a practical application. However, the scheduling literature including both multi skill and task preemption, all the more grounded in a real business application, is rather sparse [1, 34].

The presence of multiple skilled resources adds more complexity to the problem, and new types of goals aside the usual makespan and cost optimization, like assignment score and worker performance [35]. Task preemption adds another layer of complexity, and in real scenarios usually is tied to resource availability, with special focus in worker shifts and calendars. Also, it is important to note that whilst a task can be preempted, pausing its progress, maybe only part of the resources can be freed to make them available for other tasks, with others being locked in place. This has been already referred to as partial preemption [25]. Finally, practical problems also contain generalized precedence constraints and resource calendars, which have been well known from long ago. These constraints extend simple tasks precedence to detail concrete temporal conditions between tasks. Calendars on the other hand, define resource unavailability periods, hence both features make the problem harder to solve [17].

In this paper, we present a method oriented towards a real assembly line use case, requiring to include all these five features: multi skilled resources, multiple modes, (partially) preemptive tasks, calendars and generalized precedence constraints. Hence, we call our model the partially preemptive- multi-skill/mode resource-constrained project scheduling problem with generalized precedence relations and resource calendars (PP-MS-MM-RCPSP/max-cal). Our solution is aimed for industrial domains, but can solve generic problems in the scope of the model as well. We propose an extended version of the large neighborhood search (LNS) method [21, 11], softening constraints initially to find a base solution that will be improved on each iteration. We observed this approach to be faster than other more direct constraint programming approaches. We evaluated our method against different benchmarks, including one using real data from aerospace manufacturing plants, real instances from an hazardous material examination facility [25] and standard multi-skill scheduling instances [38].

The rest of the paper is structured as follows: in Section 2, literature about multi skill scheduling and preemption in scheduling is discussed, along with previous scheduling works oriented towards industrial applications. Section 3 describes the domain of our problem and contains the formulation of our model, including a new CP formulation tailored to preemption. In Section 4 we detail our approach explaining the algorithm we developed to solve PP-MS-RCPSP/max-cal. Following this, Section 5 contains the experiments and benchmarks conducted to evaluate the performance of our solution. Lastly, our conclusions can be seen in Section 6.

2 Multi-skill/mode, preemption, calendars and generalized precedence in the literature

The multi-mode RCPSP (MM-RCPSP) allows reaching a compromise between resource usage and task duration as it frequently occurs in practice. It also includes non renewable resources to model budget constraints that may prevent from using the fastest modes for all tasks. Many solution methods have been proposed over the years from local search, to sat-based methods using hyperheuristics. [6, 9, 15, 37]. CP approaches are currently highly popular and efficient tools to solve RCPSP variants [18] and in particular the MM-RCPSP [36].

MS-RCPSPs are a particular MM-RCPSP based on the concept of skills enabling the compact representation of a possibly large set of modes corresponding to combination of resources managing a set of skills required by a task. There is a variety in the methods and models to solve the problem: tree search [29], genetic algorithms [20], mixed-integer linear programming [3, 10, 19, 31]. On the standard MS-RCPSP, we will compare our method with two state-of-the-art methods: the CP approach using no-good learning proposed in [38] and the greedy randomized adaptive search procedure (GRASP) presented in [25].

Preemption can be defined as the capability of stopping the execution of a task in the generated schedule, releasing the resources it was using in order to allow for the execution of a different one, and being able to be continued later. In some cases preemption makes the problem easier to solve when an NP-hard non preemptive scheduling problem has a polynomial preemptive counterpart [5] but in the case of job-shop or resource-constrained project scheduling the problem may become harder to solve due to a much larger search space [7, 22]. Only a handful of studies have combined multi-skilled resources and preemption in the same work. We can cite [12, 24, 25, 26], where mixed-integer linear programming, constraint programming and metaheuristics methods were proposed. Prominent preemption can cause trouble to CP approaches: in [25] an experiment revealed that MILP obtained better results on a set of highly preemptive MS-RCPSP than the default search of IBM CP Optimizer. For preemptive MS-RCPSP, we will compare our method with the hybrid GRASP-LNS heuristics proposed in [26]. For their application, the authors also considered partially preemptive tasks, where only part of the resources are released during preemption. Due to industrial relevance, we also include partial preemption in our model.

Resource calendars are often unavoidable characteristics in industrial contexts. They can be linked to a basic form of preemption in the sense that an on-going task has to be preempted when one of its resource becomes unavailable due to an off-time in the calendar. In [13], calendars were used in a MS-RCPSP context without including task preemption. Our model considers both situations and tasks can be preempted or not by calendars as frequently observed [17].

Generalized precedence constraints, also considered in our model specify more complex temporal relations between tasks than standard precedence constraints. They make the problem more complex in all RCPSP variants as even finding a feasible solution becomes NP-hard in the presence of generalized precedence constraints. Many works have considered such constraints in the standard RCPSP [4, 32] and also for multi-mode preemptive RCPSP [27]. The RCPSP/max-cal problem involving both generalized precedence and calendar constraints was again successfully solved by CP in [17].

A difference between the performance of generic solvers on academic RCPSP models and on their adaptation to domain problems with very specific requirements is observed in other works and we also could experience it in our benchmark experiments: usually the more special characteristics the problem has, the worse is the performance of generic solvers. Our model falls into this category as it includes all the above-mentioned complicating characteristics. The LNS approach appears as a technique of choice to find the right compromise between genericity and performance.

3 PP-MS-MM-RCPSP/max-cal: definition and formulation

The PP-MS-MM-RCPSP/max-cal generalizes the partially preemptive MS-RCPSP introduced in [26]. An additional feature is the possibility to execute the task in different modes, in the same way it is done in multi-mode RCPSP [15]. Generalized precedence constraints and

calendars are also added. Formally a PP-MS-MM-RCPSP/max-cal is defined by:

1. $\mathcal{A}$ a set of activities to schedule, $\mathcal{A} = \{1, ..., n\}$, two special activities are created by convention: the source task 1 (predecessor of all other tasks) and sink task n (successor of all other tasks);
2. Each activity $i \in \mathcal{A}$ can be performed in m_i different modes, $m_i \in M_i = \{1, ..., |M_i|\}$; (as defined in [28])
3. $\mathcal{R}^\rho$ a set of renewable resource types, $\mathcal{R}^\rho = \{R^\rho_1, ..., R^\rho_{m^\rho}\}$. $\forall k \in \mathcal{R}^\rho, t \in \mathbb{N}, B_{kt}$ is the discrete amount of available resource k at discrete time t (available between time t and $t+1$);
4. $\mathcal{R}^\nu$ a set of non-renewable resource types, $\mathcal{R}^\nu = \{R^\nu_1, ..., R^\nu_{m^\nu}\}$; $\forall k \in \mathcal{R}^\nu$, B_k is the total capacity of the non-renewable resource;
5. $\mathcal{O}$ the set of disjunctive resources representing individual skilled workers (from now on, we will refer to them as Operators);
6. $\mathcal{L} = \{1, ..., L\}$ the set of skills;
7. $\forall o \in \mathcal{O}, t \in \mathbb{N}, A_{ot} \in \{0, 1\}$ indicates if the Operator o is present or not at time t (hence we treat the temporal availability of Operators as fixed from the problem definition);
8. $\forall o \in \mathcal{O}, l \in \mathcal{L}, y_{ol} \in \{0, 1\}$ indicates if Operator o masters skill l;
9. $\forall i \in \mathcal{A}$, r_i is the release time of task i;
10. $\forall i \in \mathcal{A}$, d_i is the deadline time of task i;
11. $\forall i \in \mathcal{A}$, p_{i,m_i} is the processing time under mode $m_i \in M_i$;
12. $\forall i \in \mathcal{A}, k \in \mathcal{R}^\rho \cup \mathcal{R}^\nu, b_{i,m_i,k}$ is a natural number representing the resource demand of activity i for resource k, under mode m_i;
13. $\forall i \in \mathcal{A}, l \in \mathcal{L}, s_{i,m_i,l}$ is a natural number representing the skill requirement of activity i under mode m_i;
14. P is the set of precedence constraints $\mathcal{A} \times \mathcal{A}$ specifying which activity should precede another one;
15. $P_{sync-start}$ is the set of activity pairs that must start at the same time;
16. $P_{startlag}$ is the set of ordered pairs (i, j) specifying $\Delta s_{(i,j)}$, the minimum time lag between start of i and start of j;
17. $P_{sync-end}$ is the set of activity pairs that must end at the same time;
18. P_{endlag} is the set of ordered pairs (i, j) specifying $\Delta e_{(i,j)}$, the minimum time lag between end of i and start of j;
19. The set $\mathcal{A}$ is split in three different subsets :
    - $\mathcal{A} = \mathcal{A}^P \cup \mathcal{A}^{NP} \cup \mathcal{A}^{PP}$;
    - $\mathcal{A}^P$ is the set of fully preemptive activities (activities can be preempted and all the resources are released);
    - A^{NP} is the set of non-preemptive activities;
    - A^{PP} is the set of partially preemptive activities (where at least one resource is not releasable);
20. $\forall i \in \mathcal{A}, m_i \in M_i, k \in \mathcal{R}^\rho, \rho_{i,m_i,k} \in \{0, 1\}$ indicates if resource k is releasable for activity i under mode m_i.

3.1 Known variants in the literature

Our generic formulation encompasses more classical scheduling problems, that can be therefore solved using the proposed solution, such as:

- the classical RCPSP ($\mathcal{L} = \emptyset$, $\mathcal{R}^\nu = \emptyset$, $\forall k \in \mathcal{R}^\rho, \forall t, t' \in \mathbb{N}, B_{kt} = B_{kt'}$, $\mathcal{O} = \emptyset$, $\mathcal{A}^P = \emptyset$, $\mathcal{A}^{PP} = \emptyset$, $\forall i \in \mathcal{A}, M_i = \{1\}$);

- the classical multi-mode RCPSP ($\mathcal{L} = \emptyset$, $\forall k \in \mathcal{R}^\rho, \forall t, t' \in \mathbb{N}, B_{kt} = B_{kt'}$, $\mathcal{O} = \emptyset$, $\mathcal{A}^P = \emptyset$, $\mathcal{A}^{PP} = \emptyset$, $\forall i \in \mathcal{A}, M_i = \{1, ...|M_i|\}$);
- the classical multi-skill RCPSP ($\mathcal{L} = \{1, ..., L\}$, $\mathcal{O} \neq \emptyset$, $\mathcal{A}^P = \emptyset$, $\mathcal{A}^{PP} = \emptyset$, $\forall i \in \mathcal{A}, M_i = \{1, ...|M_i|\}$).

3.2 Constraint Programming formulation

We developed a combinatorial optimisation library containing the CP model for PP-MS-MM-RCPSP/max-cal. It can be found as part of an open source framework to solve discrete optimization problems [1].

A big focus of the library is on the RCPSP's class of problems described in this paper. All solver approaches, from Local search, rule based heuristics, CP, LP and LNS methods are either coded using the discrete-optimisation library or wrapped into it, allowing the user to easily benchmark methods and hybridize different approaches.

The following sections detail the decision variables and constraints of our formulation for PP-MS-MM-RCPSP/max-cal.

3.2.1 Decision variables

We assume that each task can be preempted at regular discretized time points, resulting in a sequence of small non-preemptive chunks for each activity. A subpart of an activity is thus defined as a subset of adjacent chunks of this activity. Let's define $max_{preemption}$ as an arbitrary input of our problem which represents an upper bound on the number of preemption breaking times allowed for all our activities. We will note $\mathcal{J} = \{1, .., max_{preemption}\}$ the set of preemption breaking time indexes and $\mathcal{J}^- = \{2, .., max_{preemption}\}$

1. Starting time decision : $starts$ is a $|\mathcal{A}| \times max_{preemption}$ matrix, which contains the starting time of subparts of all the tasks in increasing order.
2. Duration decision : $durations$ is a $|\mathcal{A}| \times max_{preemption}$ matrix, which contains the duration of subparts of all the tasks.
3. Mode decision : $modes$ is a $|\mathcal{A}|$ vector specifying in which mode a task is executed. $\forall i \in \mathcal{A}, modes[i] \in M_i$.
4. Resource allocation : $allocation$ is a $|\mathcal{O}| \times |\mathcal{A}| \times max_{preemption}$ 3D binary matrix which indicates which worker o is allocated to each subpart of an activity.

3.2.2 Constraints

We will use the notation [:] in matrix indexing to ease the readability of constraints. For e.g $starts[i, :]$ is the vector of starting times for the task i and $allocations[o, :, :]$ is the $|\mathcal{A}| \times max_{preemption}$ matrix allocation of resource o to all activity subparts.

1. Resource consumption constraint :
$$\forall k \in \mathcal{R}^\rho, cumulative(starts, durations, b_{modes,k}, B_{k:})$$
where $b_{modes,k}$ is an array of dimension $|\mathcal{A}| \times max_{preemption}$ defined by:
$\forall i \in \mathcal{A}, j \in \mathcal{J}, b_{modes,k}[i, j] = b_{i,modes[i],k}$, being $b_{modes,k}[i, j]$ the resource demand of activity i on the j-th subpart of its execution, which doesn't depend on j in our problem. We use the global cumulative constraint implemented in most CP language to model the cumulative resource consumption in RCPSPs [30].

[1] `https://github.com/airbus/discrete-optimization`

To take into account variable resource availability, we use a discrete stepped function $max_t(B_{kt}) - B_{kt_{step}}$ at each t_{step} that includes an artificial fixed task consuming it. This way, whenever availability changes, that task removes the resources from the pool during a period defined by the step function. In our domain, variable resource availability translates into worker's shifts and calendars, also indirectly defining points where a task needs to be preempted. For simplicity we didn't include it in the description of the constraint.

2. Non-renewable resources:
$\forall k \in \mathcal{R}^{\nu}, \sum_{i \in \mathcal{A}} b_{i,modes[i],k} \leq B_k$. This non-renewable resource constraint is only meaningful when there are several possible modes. In multi-mode settings, some *modes* value wouldn't satisfy the constraint.
3. Skills requirement:
$\forall i \in \mathcal{A}, j \in \mathcal{J}, l \in \mathcal{L}$,
$durations[i,j] > 0 \rightarrow \sum_{o \in \mathcal{O}} allocation[o,i,j] \cdot y_{ol} \geq s_{i,modes[i],l}$
Contrary to other multi-skill variants, we consider that Operators assigned to a task contribute to the task skill requirements with all of their skills, similar to what it is seen in [26].
4. Operator availability:
$$\forall o \in \mathcal{O}, cumulative(starts, durations, allocation[o,:,:], A_{o:})$$
Variable availability of an Operator is dealt with in the same way as Constraint 1 above.
5. Precedence relation: $\forall (i,j) \in P$,
$starts[j,1] \geq starts[i, max_{preemption}] + durations[i, max_{preemption}]$
6. $P_{sync-start}$ relations:
$\forall (i,j) \in P, starts[i,1] = starts[j,1]$
7. $P_{startlag}$ relations:
$\forall (i,j) \in P_{startlag}, starts[j,1] \geq starts[i,1] + \Delta s_{(i,j)}$
8. $P_{sync-end}$ relations: $\forall (i,j) \in P_{sync-end}$,
$starts[j,1] = starts[i, max_{preemption}] + durations[i, max_{preemption}]$
9. P_{endlag} relations:
$\forall (i,j) \in P_{endlag}$,
$starts[j,1] \geq starts[i, max_{preemption}] + durations[i, max_{preemption}] + \Delta e_{(i,j)}$
10. Tasks duration :
    a. $\forall i \in \mathcal{A}^{NP}, durations[i,1] = p_{i,modes[i]} \wedge (\forall j \in [2, max_{preemption}], durations[i,j] = 0)$
    b. $\forall i \in \mathcal{A} \setminus \mathcal{A}^{NP}, \sum_{j \in \mathcal{J}} durations[i,j] = p_{i,modes[i]}$
11. Release time:
$\forall i \in \mathcal{A}, starts[i,1] \geq r_i$
12. Deadline time:
$\forall i \in \mathcal{A}, starts[i, max_{preemption}] + durations[i, max_{preemption}] \leq d_i$
13. Conventions constraints for *starts* and *durations*:
The following constraints are modeling choices aiming to help the solver to explore the search space.
    a. $\forall i \in \mathcal{A}, j \in \mathcal{J}^{-}, starts[i,j] \geq starts[i,j-1] + durations[i,j]$: precedence constraints between each subparts of the task.
    b. $\forall i \in \mathcal{A}, j \in [1, max_{preemption} - 1], durations[i,j] = 0 \rightarrow durations[i,j+1] = 0$: As soon as there is a 0 duration subtask, all the following ones are 0 too.
    c. $\forall i \in \mathcal{A}, j \in [1, max_{preemption}], durations[i,j] = 0 \rightarrow starts[i,j] = starts[i,j-1] + durations[i,j-1]$: Whenever there is a 0 duration, the remaining *starts* values are uniquely determined by previous values, pruning redundant solutions.

14. Partially preemptive and non releasable resources :
We introduce variable $blockedduration[i,j,k]$, $\forall i \in \mathcal{A}$, $j \in \mathcal{J}$, $k \in \mathcal{R}^{\rho}$ as the usage duration of resource k for task i and activity subpart j. It is regulated by the following constraint: $\forall j \in \mathcal{J}$,
$\rho_{i,modes[i],k} = 1 \rightarrow blockedduration[i,j,k] = durations[i,j]$
This means if the resource is releasable then the duration of the usage is the same as the duration of the subtask.
On the other hand, to describe the consumption of a resource over the total span of the activity for not releasable resources, we have:
$\rho_{i,modes[i],k} = 0 \rightarrow (blockedduration[i,1,k] = starts[i, max_{preemption}] - starts[i,1]) \wedge$
$(blockedduration[i,j,k] = 0, \forall j \in \mathcal{J}^{-})$
This will set $blockedduration[i,1,k]$ to the total span of the activity, including preempted time, and ignore the other subparts.
Then, when indicating the *cumulative* constraint, instead of the normal duration, this resource uses $blockedduration$:

$$\forall k \in \mathcal{R}^{\rho}, cumulative(starts, blockeduration, b_{modes,k}, B_{k:})$$

We note that Constraint (13) implementing preemption won't always lead to a feasible solution without backtracking even if there are no time windows or maximum time lags when using CP-Optimizer solver.

Let us take a simple example of a single preemptive activity A_1 of duration 5 task decomposed in 3 subtasks $A_{1,1}$, $A_{1,2}$ and $A_{1,3}$. Suppose the solver sets the start time of $A_{1,1}$ to 0 and the end time of $A_{1,1}$ to 3 (and consequently its duration to 3). Suppose now that the solver sets the start time and the end time of $A_{1,2}$ to 3 (duration 0). Then according to constraint (13b) the duration of $A_{1,3}$ is set to 0 and a failure occurs due to the impossibility to satisfy constraint (10b).

To deal with this issue, we took advantage of the expressiveness of CP-Optimizer to add an alternative variant in our model: using the concept of interval instead of using variables *starts* and *durations*. An interval is defined as follows: $\forall i \in \mathcal{A}, interval[i]$ is a $|max_{preemption}|$ array of optional intervals variable. Each interval has the attributes *present* (boolean), *duration*, *start*, *end* (all integers). We also create one unique interval for each task, called spantask. New constraints are applied to this *interval* variable :

15. $\forall i \in \mathcal{A}$, $s.t\ p_{i,modes[i]} >= 1, \forall j \in \mathcal{J}, interval[i,j].duration \geq 1$: We don't consider 0 duration for task intervals, which avoids the above-presented issue. This could also help the solver to remove unnecessary solutions in its search space since otherwise there could be multiple equivalent solutions.

16. $\forall i \in \mathcal{A}, j \in \mathcal{J}^{-}, interval[i,j].start \geq interval[i,j-1].end$: precedence constraints between sub-intervals of the same task.

17. $\forall i \in \mathcal{A}, j \in [1, max_{preemption} - 1], interval[i,j].present = False \rightarrow interval[i,j+1].present = False$: When one sub-interval is not present, the remaining ones are not present either. This is equivalent to 13b constraint.

18. $\forall i \in \mathcal{A}, \sum_{j \in \mathcal{J}} (interval[i,j].duration * interval[i,j].present) = p_{i,modes[i]}$ The sum of duration of present intervals should sum to the duration of the task.

19. $\forall i \in \mathcal{A}, span(\text{spantask}[i], interval[i,:])$: We use the native constraint *span* of CPOptimizer so that the spantask$[i]$ spans over all present intervals in $interval[i]$, ignoring the non-present.

The precedence constraints are written using the spantask interval. We call this new model specific to CPOptimizer, Model *CP-SmartPreemption* while Model (1–14) is called *CP-Base*.

3.2.3 Objective function

We will focus exclusively on the makespan, which in terms of our formulation is equal to the ending time of the sink task n. Hence the goal is to minimize

$$makespan = starts[n, max_{preemption}] + durations[n, max_{preemption}]$$

However as detailed in section 4.3, due to the difficulty to obtain feasible solution, in our solution approach we opt in a first phase for a relaxation of a set of constraints and a penalization of their violation in the objective function. Thus the actual objective in this phase is a lexicographic optimization of the violation penalty and the makespan (see section 4.3).

3.3 A small PP-MS-MM-RCPSP/max-cal instance and its optimal solution

We provide an example PP-MS-MM-RCPSP/max-cal instance to illustrate its output from a simple problem definition contained in Table 1. It contains 6 activities to schedule (including source and sink tasks); it also includes multi-mode tasks, variable operator availability, release and deadlines, complete and partial preemption and finally one synchronisation constraint. This instance can be found in the toy model folder of our open source model's repository [2]. The optimal solution is depicted in Figure 1. We can observe that the tasks A_1 and A_3 are partially preemptive activities : the resource R_1 is therefore still used even though the task is paused. The calendar constraint of Operator 1 is visible in the corresponding Gantt chart where we see no operations assigned to it during its break time. The mode allocation is the following : $mode_{A_0} = 1, mode_{A_1} = 2, mode_{A_2} = 1, mode_{A_3} = 2, mode_{A_4} = 2, mode_{A_5} = 1$.

Table 1 An instance of PP-MS-MM-RCPSP/max-cal.

Activity	Mode	Duration	Skills	Resource	Deadline	Release	Preemption	Successors
A_0	1	0	-	-	-	-	-	A_1, A_2, A_3, A_4, A_5
A_1	1	5	l_1	R_1	-	-	$\mathcal{A}^{PP}$	A_5
	2	3	l_1	R_1	5	-	$\mathcal{A}^{PP}$	A_5
A_2	1	1	l_3, l_4	$(R_1, 1)$	-	2	$\mathcal{A}^{NP}$	A_3, A_5
	2	2	l_3	$(R_1, 1)$	-	2	$\mathcal{A}^{NP}$	A_5
A_3	1	3	l_2	$(R_1, 1)$	-	-	$\mathcal{A}^{PP}$	A_5
	2	2	l_3, l_2	$(R_1, 1)$	-	-	$\mathcal{A}^{PP}$	A_5
A_4	1	2	l_3	-	-	5	$\mathcal{A}^{P}$	A_5
	2	3	-	$(R_1, 1)$	-	5	$\mathcal{A}^{P}$	A_5
A_5	1	0	-	-	-	-	-	-

Operator	Skills	Calendar
O_1	l_1, l_3	$[1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0]$
O_2	l_1, l_2, l_4	$[1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1]$

Resource	Capacity
R_1	2

$P_{sync-start}$	$[(A_3, A_4)]$

[2] https://github.com/g-poveda/do_experiments

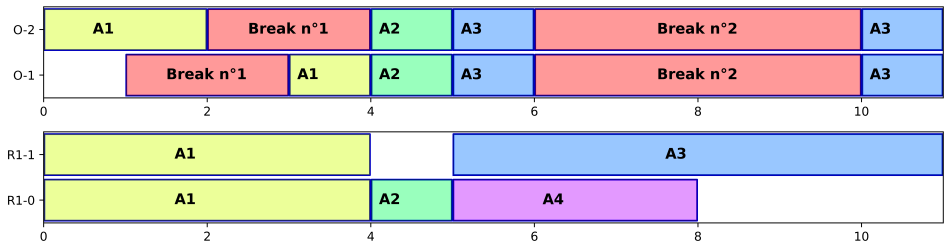


Figure 1 Operator and resource oriented Gantt chart for the example instance.

4 Algorithms

4.1 Generic large neighborhood search algorithm

Our approach to solve the PP-MS-MM-RCPSP/max-cal is to use a generalisation of Large Neighborhood Search (LNS) for scheduling problems [21]. In LNS, We iteratively improve the solution quality of a Master Problem $\mathcal{MP}$ by solving at each step a Reduced Master Problem $\mathcal{RMP}$, based on the $\mathcal{MP}$ and previously found solutions. The way $\mathcal{RMP}$ are built are the core of LNS methods. The generic LNS algorithm is described in Algorithm 1, where X^{iter} denotes the solution at iteration $iter$ and Y^{iter} its objective value.

Algorithm 1 Generic Large Neighborhood Search Algorithm.

Begin
1: $Y^* = \infty, X^* = None$
2: $(X^0, Y^0) = (X^*, Y^*) = initial_{solution}(P)$
3: $iter = 0$
4: **repeat**
5: $\quad \mathcal{RMP} = buildsubproblem(\mathcal{MP}, X^{iter})$
6: $\quad X^{iter+1}, Y^{iter+1} = solve(\mathcal{RMP})$
7: $\quad$ **if** $Y^{iter+1} \leq Y^*$ **then**
8: $\quad\quad X^* \leftarrow X^{iter+1}$
9: $\quad\quad Y^* \leftarrow Y^{iter+1}$
10: $\quad iter \leftarrow iter + 1$
11: **until** stop criterion is met
12: **return** X^*, Y^*

4.2 Application to the PP-MS-MM-RCPSP/max-cal

4.2.1 Initial solution provider

We rely on a generalisation of the serial schedule generation scheme (SGS) procedure [16] to produce an initial solution for PP-MS-MM-RCPSP/max-cal (Algorithm 1, line 2). A similar generalisation was implemented in [26] being the closest one we found in the literature since it includes multi-skill, preemption and partially releasable resources.

Our SGS implementation takes as input a priority ordering of activities $order\_act$, given as a permutation of $\mathcal{A}$ and, for each activity $i \in \mathcal{A}$, another priority ordering $order\_res_i \in S_{|\mathcal{O}|}$ of the workers $o \in \mathcal{O}$ to be assigned to the activity and the mode id $modes_i$ chosen to run the activity, as defined for the CP model.

Activities are scheduled incrementally based on their priority as in classic serial SGS, at their earliest possible start date considering resource availability. Worker allocation is also done greedily using the *order_res* array. The preemptivity feature of our problem allows us to schedule subpart of some activities. Our open source library includes an implementation module handling all the features of PP-MS-MM-RCPSP/max-cal except for the $P_{sync-start/end}$ constraints, release and deadline constraints. The greedy procedure doesn't ensure that the constraints are feasible for these hard constraints.

Using the *SGS* procedure makes possible to run simple local search algorithms to get one initial solution of the problem. Such solver explores the vectored state space $order\_act, order\_res, modes$, and evaluate its fitness using *SGS*. The output evaluation of *SGS* is the same than the LNS or CP ones : makespan + potential penalty when deadlines and other constraints are not fulfilled (see details in Section 4.3). The state space being a permutation one, we are using moves chosen randomly from a portfolio of potential moves : partial and total shuffling, swap 2 random activities, 2-opt moves. Only one specific mutation has been also implemented to correct potential deadlines constraints (that put first the late task). In this paper we are using simulated annealing (SA) [14] as a local search solver. SA was parameterized with an initial temperature of $T = 3$ and exponential cooling schedule of factor $\alpha = 0.999$. A restart strategy also ensures to roll back to the current best solution when we haven't improved the quality after $N = 300$ iterations.

4.2.2 Subproblem building

To build the reduced master problem, we rely on the constraint programming formulation and include additional constraints, which makes it simpler to solve. We decompose the subproblem building procedure (Algorithm 1, line 5) in two main methods. The first one, *activityselector* will split the set of activities $\mathcal{A}$ into 2 disjunctive subsets $\mathcal{A}_{sub}$ and $\mathcal{A}_{\overline{sub}}$. Different methods have been implemented and tested to select $\mathcal{A}_{sub}$ and $\mathcal{A}_{\overline{sub}}$ as described below:

1. Random selection : given $f \in \mathbb{R}, 0 \leq f \leq 1$,
   $\mathcal{A}_{sub} = sample(\mathcal{A}, \lfloor f \cdot |\mathcal{A}| \rfloor)$, and $\mathcal{A}_{\overline{sub}} = \mathcal{A} \setminus \mathcal{A}_{sub}$
2. Random selection and neighbors : consist in the previous selection, then add the predecessors of each task in $\mathcal{A}_{sub}$. Formally, given $f \in \mathbb{R}, 0 \leq f \leq 1$:
   $\mathcal{A}_{sub,1} = sample(\mathcal{A}, \lfloor f \cdot |\mathcal{A}| \rfloor)$, and
   $\mathcal{A}_{sub} = \mathcal{A}_{sub,1} \cup \{j \in \mathcal{A}\, s.t\, \exists (x, y) \in P,\, x = j \,\wedge y \in \mathcal{A}_{sub,1}\}$
3. Cut in equal parts : given $c \in \mathbb{N}$, we sort the activities by increasing order of starting times in the current best solution X^*, then cut this ordered list in c equal pieces. When called, the function returns one of these c parts. A good strategy we observed consists in returning them in increasing order.
4. Generalized precedence adapted selection : when considering the constraints of deadlines, release dates, and the generalized precedence constraints ($P_{sync-start}$, $P_{startlag}$, $P_{sync-end}$, P_{endlag}) , we can reach a solution X^* violating some of the constraints. Therefore we add in priority the tasks responsible for constraints violation in $\mathcal{A}_{sub}$ and their predecessors in the precedence graph.
5. Mixing methods : given a pool containing the previous four methods, it will return $\mathcal{A}_{sub}$ and $\mathcal{A}_{\overline{sub}}$ of one of the pool members (either randomly at each iteration, or based greedily on the effectiveness of the selected method based on previous iterations).

Once we have the *activityselector* function, its output ($\mathcal{A}_{sub}$ and $\mathcal{A}_{\overline{sub}}$) can be used in the *constraintbuilder* function to add constraints to the CP model as described in Algorithm 2. Instead of totally locking the variables corresponding to $\mathcal{A}_{\overline{sub}}$ and keeping free for optimisation

the variables of $\mathcal{A}_{sub}$, we consider different ϵ values to constrain start and duration variables. We typically assign high values to parameters $\epsilon_{sub,*}$ and small values to $\epsilon_{\overline{sub},*}$.

Algorithm 2 Constraint builder procedure.

$constraintbuilder(\mathcal{A}_{sub}, \mathcal{A}_{\overline{sub}}, X, model_{cp}, params_{constraints})$:
$\epsilon_{sub,+}, \epsilon_{sub,-}, \epsilon_{\overline{sub},+}, \epsilon_{\overline{sub},-}$,
$\epsilon dur_{sub,+}, \epsilon dur_{sub,-}, \epsilon dur_{\overline{sub},+}, \epsilon dur_{\overline{sub},-}, = params_{constraints}$
for all $i \in \mathcal{A}$ **do**
  for all $j \in [1, max_{preemption}]$ **do**
    if $X.durations[i,j] > 0$ **then**
      if $i \in \mathcal{A}_{sub}$ **then**
        $(\epsilon_-, \epsilon_+) \leftarrow (\epsilon_{sub,-}, \epsilon_{sub,+})$
        $(\epsilon dur_-, \epsilon dur_+) \leftarrow (\epsilon dur_{sub,-}, \epsilon dur_{sub,+})$
      else if $i \in \mathcal{A}_{\overline{sub}}$ **then**
        $(\epsilon_-, \epsilon_+) \leftarrow (\epsilon_{\overline{sub},-}, \epsilon_{\overline{sub},+})$
        $(\epsilon dur_-, \epsilon dur_+) \leftarrow (\epsilon dur_{\overline{sub},-}, \epsilon dur_{\overline{sub},+})$
      $model_{cp} \leftarrow X.starts[i,j] - \epsilon_- \leq model_{cp}.starts[i,j] \leq X.starts[i,j] + \epsilon_+$
      $model_{cp} \leftarrow max(0, X.durations[i,j] - \epsilon dur_-) \leq model_{cp}.durations[i,j] \leq X.durations[i,j] + \epsilon dur_+$

4.2.3 Subproblem solving

To solve the subproblem we use the open source lazy clause generation solver Chuffed[3] or alternatively IBM's CP-Optimizer[4] backend. Both rely on a CP formulation done in Minizinc language, and worked well in our experiments. At each iteration of the solver, we set an upper time computation h. In case we want to optimize The objective function can be the final activity n's end or the end of the subset of activities $\mathcal{A}_{sub}$ we consider the most important in the $\mathcal{RMP}$. Both strategies impact slightly the convergence of the overall algorithm but we didn't assessed the details in our current experiments.

After retrieving the solution of the $\mathcal{RMP}$ from the solver, it is post-processed by a left shifting procedure that compresses the full schedule. It is particularly useful in the case where the constraints introduced in *constraintbuilder* create idle times in the resulting schedule.

4.3 Relaxing constraints

The current *SGS* implementation does not return a feasible solution when certain constraints of PP-MS-MM-RCPSP/max-cal are present. Namely, those are the deadline and generalized precedence constraints $P_{sync-start}$, $P_{sync-end}$, $P_{startlag}$, P_{endlag}. By relaxing constraints (6), (7), (8), (9), (12) from Subsection 3.2.2 we include a violation penalty into the objective function. This violation penalty for $P_{sync-end}$ is illustrated in Figure 2. The objective function that LNS and underlying CP solver minimizes is $makespan + M \cdot penalty$ where M is a large number.

This means that in practice, there will be 2 phases of optimisation :

- Feasibility phase : In this phase the violation penalty decreases, converging to 0.
- Optimisation phase : the algorithm optimises the original objective function ($makespan$)

[3] https://github.com/chuffed/chuffed
[4] https://www.ibm.com/analytics/cplex-cp-optimizer

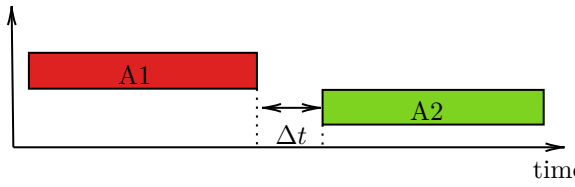


Figure 2 Δt is added to the objective function whenever $(A1, A2) \in P_{sync-end}$.

5 Experimental results

5.1 Manufacturing domain instances

We ran a series of experiments in order to evaluate our LNS method for PP-MS-MM-RCPSP/max-cal. All experiments have been done in a MacBook Pro with 2,7 GHz Quad-Core Intel Core i7. In the first experiment, our goal is to assess the performance of our method in a practical situation, so we extracted data samples from a real manufacturing use case. The experiment relies on two scheduling problem base instances (A and B) obtained from one of the assembly plants at Airbus, and are described in Table 2. Using these 2 base instances, we built 32 different instances, using all the possible combinations of the following features :

- Calendar: we can use our definition for resource consumption (Constraint 1 in section 3.2) or consider complete resource availability at any moment, hence making vectors $B_k : \forall k \in \mathcal{R}^\rho$ always constant and $A_o : \forall o \in \mathcal{O}$ always 1.
- Temporal Constraints: we include or not the deadline times, release times (described in Section 3.2 as the constraints number 11 and 12) and generalized precedence constraints (constraints number 6 to 9)
- Preemptive : tasks can be preempted if needed or preemption is forbidden; in the non preemptive use case, $\mathcal{A}^{NP} = \mathcal{A}$
- Multiskill : skilled workers are present or not; in the non multiskill use case, workers belong to $\mathcal{R}$ and $\mathcal{O} = \emptyset$

Table 2 Base instance description of the manufacturing use case.

Instance	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{L}\rvert$	$\lvert\mathcal{O}\rvert$	$\lvert P\rvert$	$\lvert P_{ss}\rvert$	$\lvert P_{se}\rvert$	$\lvert P_{sl}\rvert$	$\lvert P_{el}\rvert$
A	291	8	23	27	842	3	70	0	4
B	199	3	6	6	676	1	34	6	4

We ran our LNS algorithm with a time limit of 1800 seconds for all of the instances. As a comparison, we also ran the simulated annealing algorithm (SA) and a direct CP solving of Model *CP-Base* using Minizinc and the CP-Optimizer backend, the Chuffed backend and the *CP-SmartPreemption* model for the same amount of time.

The results for each instance are detailed in Table 3. Four of those instances (number 8 to 11) are infeasible by design, because they include the calendar constraints but not the preemption ones, so we left out those and will focus on the remaining 28 instances for our experiments. For the sake of clarity, we only show *CP-SmartPreemption* results (CP-S in the table) as it it consistenly better than the *CP-Base* model whether the latter is solved by Chuffed or CP-Optimizer. A comparison between *CP-SmartPreemption* and *CP-Base* can be found in Appendix A.1.

Table 3 Makespan results obtained for the 28 testing instances (splitted in two tables) using the three solvers: LNS, *CP-SmartPreemption* (abreviated as CP-S) and SA. For SA penalty values are included when the solution is not satisfying all the constraints (solutions for LNS and *CP-SmartPreemption* got no penalty). The table also includes each instance base problem (A or B) and its features: **M** for Multi skill, **G** for Generalized precedence constraints, **P** for (partial) Preemption and **C** for resource Calendars. In **bold** best results among the three algorithms.

ID	Features	LNS	CP-S	SA	p-SA	ID	Features	LNS	CP-S	SA	p-SA
0	A	**1080**	**1080**	**1080**	0	18	B G	**3053**	**3053**	3275	2171
1	A M	**1080**	**1080**	**1080**	0	19	B G M	**3109**	**3109**	3530	2590
2	A G	**6534**	**6534**	6817	128	20	B P	2340	2340	**2339**	0
3	A G M	6553	**6534**	6543	104	21	B M P	**2340**	2390	**2340**	0
4	A P	**1080**	**1080**	**1080**	0	22	B G P	3053	**3044**	3148	750
5	A M P	**1080**	2495	**1080**	0	23	B G M P	**3117**	3180	3456	714
6	A G P	**6534**	**6534**	**6534**	0	24	B C	3202	**3196**	3250	0
7	A G M P	6553	6612	**6537**	0	25	B C M	3289	**3208**	3263	0
12	A C P	**7154**	-	**7154**	0	26	B C G	**4949**	**4949**	5096	4727
13	A C M	**7154**	-	**7154**	0	27	B C G M	5096	**4949**	5096	4359
14	A C G P	7866	-	**7777**	0	28	B C P	3205	3195	**3192**	0
15	A C G M P	**7777**	-	7805	0	29	B C M P	3193	3208	**3192**	0
16	B	2340	**2339**	2373	0	30	B C G P	4580	**4383**	4436	1167
17	B M	2350	**2346**	2375	0	31	B C G M P	**5010**	-	4840	1132

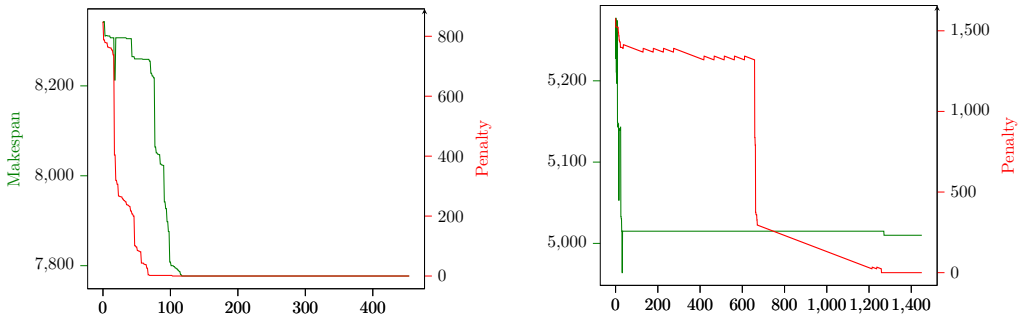
As we can see, LNS reaches the best results on 15 out of the 28 instances, *CP-SmartPreemption* on 16 and SA on 13. By comparing the different instances where each method excels, we observe that instances with high complexity in the type of constraints, i.e a combination of having generalized precedence constraints, calendar, multiskill or preemptive tasks are the ones for which we would rely the most on LNS: instance 15 and 31 are indeed the ones including all the available features. For those instances, *CP-SmartPreemption* is not returning any result in the allowed time whereas SA fails to find feasible solutions.

On the other hand, SA is generally competitive as long as there is no generalized precedence constraint involved. But if those are present it finds difficulty in yielding a solution, which was expected since the *SGS* does not fulfill all the generalized precedence constraints. However, in some of the instances SA got better results with some of those features present, thus we theorize that the solution space heavily impacts in this method, helping to overcome the presence of complex constraints and obtaining nonetheless an optimal solution: if we do not take into account instances where no feasible solution was found, SA achieves best results in 13 of 20 instances.

Figures 3a and 3b show makespan and violation penalty from the solutions found using our method for the most complex two variants (containing all the possible features). We can confirm that our strategy of relaxing certain constraints at the cost of adding a penalty is useful to get an initial solution that will be improved from that point, as described in Subsection 4.3.

5.2 Experiments on instances with multiskill and partial preemption

We conducted a second experiment using instances of partial preemptive multi-skill problems based on the schedule of operations of a nuclear facility [26]. The benchmark is divided in 4 different sets, each having different distribution combinations of $\mathcal{A}^P, \mathcal{A}^{NP}$ and $\mathcal{A}^{PP}$.



(a) Instance n°15. **(b)** Instance n°31.

Figure 3 Evolution of solutions found for LNS algorithm in two of the instances. Makespan shows in green at the left Y axis and penalty in red at the right Y axis for both of the instances.

Table 4 Average gap to lower bound for different algorithms and by set of instances. Best results for each set are bold and second best are underlined.

Algorithm	Set A	Set B	Set C	Set D
GRASP + LNS [26]	**1.56%**	**1.79%**	6.68%	**2.06%**
SA (300 seconds)	<u>1.59%</u>	<u>1.82%</u>	**5.65%**	2.33%
SA (240 s)	1.82%	1.83%	6.57%	2.17%
SA (240 s)+LNS (60 s)	1.82%	1.83%	<u>6.49%</u>	<u>2.12%</u>

The multi-skill variant of this benchmark can be directly mapped into our formalisation, except for the minimum number of operators required by activity constraints. To adapt our model we introduced in the problem instances a dummy skill s_{all} mastered by all operators. We then set skill requirement of skill s_{all} for each activity, to the minimum number of operators required.

The comparison results, detailed in Table 4, are based on the average gap to lower bound, like it was done in the original paper. We compared three of our solving methods to the one tested previously. The first one is obtained by running simulated annealing for 300 seconds, working on the permutation of tasks and the SGS algorithm. In the second we ran simulated annealing alone for 240 seconds as second baseline, and finally we ran LNS during 60 seconds. The SGS method combined with SA is competitive with the best dedicated algorithm GRASP+LNS [26]. LNS only improves slightly SA results as it can be seen when comparing SA (240 s) and SA (240 s) + LNS (60 s), which may be explained by the performance of SA itself, which is close or even better than the baseline.

5.3 Experimental results of LNS on standard multi-skill instances from the literature

Additionally, we did another benchmark comparison using the instances contained in [38]. The goal of running this second experiment is to have again comparative results to the work of [26], which our method can be related with, since it also aims at handling multi-skill and partial preemption in scheduling problems. The mentioned instances are divided in five sets, but we decided only to tests in one of them, set 1B, since it is the set with the lowest solution rates (only 12.5% solved) using the reference solvers.

Set 1B was proposed by [2] and it contains instances with 42 activities, 4 skills and between 20 to 60 resources. For these experiments, we relied on the CP model used in [38] [5], which models classical multi-skill where operator perform at most 1 skill on a task. We used this CP model in our generic Large Neighborhood Search algorithm to evaluate how well the approach generalizes to different scheduling problem variants. Baseline solutions we compare with are the ones found by running Chuffed solver with the best search strategy. They can also be found in the repository.

The current implementation of SGS prevents us to test simulated annealing as a solver or initial solution provider, because of the constraint stating that an allocated worker can at most perform 1 skill on a task. Therefore, the initial solution used in LNS is computed by using the Chuffed solver directly on the minizinc model from [38] for 30 seconds. Then, if the solution is not optimal, the large neighborhood search is used with the mixing neighborhood strategy for a maximum time of 500 s. Worker allocation is fixed only in 10% of the tasks, which empirically showed good performance in terms of improvement rate during the LNS algorithm. We are improving 151 out of 216 of the Set 1B instances (69.9% of the total) whereas Young *et al.* results [38] are still better on 29 instances (13.4%), and equal on 36 instances (17.7%). On the 151 improved instances, the average improvement of our solution over the baseline is 4.61%. On the 29 that Young *et al.* CP method [38] was better, the makespan was worsen in average by 3.46%. In total average, our method gave a 2.76% improvement on the makespan.

Finally, we conducted some preliminary experiments on the new MSLIB [34] benchmark. The authors provided us some baseline best known solutions found by a genetic algorithm (GA) approach [33] on the hardest set of instances MSLIB4. Precisely we ran experiments on the first 404 instances of MSLIB4 using CP-Chuffed 30 seconds as initial solution followed by LNS for $100s$. Our computation time is an order of magnitude higher than the GA that has a CPU time of around $15s$.

The results indicate a strong impact of the SS (skill strength) parameter of the instances. This ratio roughly gives the scarceness of the skills in the sense that $SS = 0$ means that the number of resources that master a skill is equal to the maximum demand in operators mastering this skills over the activities, i.e. the minimal value that ensure feasibility. LNS systematically outperforms GA when $SS > 0$ (90 instances) with an a average improvement of 2.9% but when $SS = 0$ (164 instances), LNS is largely outperformed by GA with an average 19.2% gap. Additional material on these results are presented in Appendix A.3.

5.4 Implementation

We provide an open repository containing scripts to rerun the benchmarks presented in Sections 5.2 and 5.3 to ease replication for further research[6]. Every multiskill variant presented in the paper is using the same python object placeholder with different attribute values. About the algorithm most of it is reused by all variants :

1. The SGS procedure is the same whatever for all benchmark and therefore the metaheuristics methods that only rely on the SGS procedure are reused identically
2. LNS algorithm 1 is the same for all benchmarks
3. Activity selectors mixing method described in 4.2 are used on all benchmarks. Experiments that justify the choice of the mixing method are presented in Appendix A.2.

[5] https://github.com/youngkd/MSPSP-InstLib/blob/master/models/mspsp.mzn
[6] https://github.com/g-poveda/do_experiments

4. Specific : We are using 2 different minizinc modeling in our benchmark, one containing the most property of PP-MS-MM-RCPSP/max-cal that is used in benchmark 5.1 and 5.2 and one another model for classical MSRCPSP for benchmark 5.3.

6 Conclusions

In this paper we have presented a large neighborhood search (LNS) method to solve partially preemptive multi-skill/mode resource-constrained project scheduling problem with generalized precedence relations and resource calendars (PP-MS-MM-RCPSP/ max-cal). Solvers including all of those features are scarce, but are usually needed to approach real manufacturing or assembly environments.

In order to validate our method, we performed three experiments to benchmark it against constraint programming and simulated annealing. In the first experiment we used data from an Airbus use case, containing a real world scenario with 32 different variations. To make a fair comparison, we designed the *CP-SmartPreemption* model that handles preemption in a clever way significantly outperforming the our baseline CP model *CP-Base*. Despite its good performances, *CP-SmartPreemption* is unable to find a feasible solution in 6 industrial instances, while the LNS method finds a solution on all instances. From the observations of the samples, we conjecture that our method works better when (almost) all complex features are present in the problem definition (i.e.: calendar, generalized precedence constraints, multiskill and preemption). Then, we tested our method using a benchmark from research work on partially preemptive multi-skill scheduling [26] to evaluate its performance in another real scenario. In this experiment, the simulated annealing methods appears competitive, slightly improved by LNS. Our method was the second best among the tested ones given the benchmark conditions, still being competitive enough to be close to the best one of that benchmark. In the last experiment on standard MS-RCPSP instances, we used our method as a mean of improvement for the solutions given by the CP based method. On the instances from [38] we improved the best known solutions in 69,9% of the instances. On the instances from [34] the performance of our approach is highly correlated with skill scarceness, which open a path for future research.

In conclusion, we found our LNS based method, available in a new discrete optimization open-source library, appropriate to solve scheduling problems containing combinations of complex features like the ones found in industrial instances, and is still reliable to be used for more academic problems.

An interesting perspective is to be able to reuse the subproblem solving information from one iteration the other to save computational time. Current Minizinc implementation has a limitation w.r.t. incrementality. Nevertheless, it would be worth to investigate how solution-based phase saving approaches [8] use nogoods and see if it is applicable in our LNS framework for future work. Adapting propagation guided LNS[23] to our framework is also a promising research direction.

References

1 Behrouz Afshar-Nadjafi. Multi-skilling in scheduling problems: A review on models, methods and applications. *Computers & Industrial Engineering*, 151:107004, 2021.

2 Bernardo F Almeida, Isabel Correia, and Francisco Saldanha-da Gama. Priority-based heuristics for the multi-skill resource constrained project scheduling problem. *Expert Systems with Applications*, 57:91–103, 2016.

3 Bernardo F Almeida, Isabel Correia, and Francisco Saldanha-da Gama. 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, 2019.

4 Lucio Bianco and Massimiliano Caramia. An exact algorithm to minimize the makespan in project scheduling with scarce resources and generalized precedence relations. *European Journal of Operational Research*, 219(1):73–85, 2012.

5 P. Brucker. *Complex Scheduling*. Springer, 1999.

6 José Coelho and Mario Vanhoucke. Multi-mode resource-constrained project scheduling using rcpsp and sat solvers. *European Journal of Operational Research*, 213(1):73–82, 2011.

7 Erik L Demeulemeester and Willy S Herroelen. An efficient optimal solution procedure for the preemptive resource-constrained project scheduling problem. *European Journal of Operational Research*, 90(2):334–348, 1996.

8 Emir Demirović, Geoffrey Chu, and Peter J Stuckey. Solution-based phase saving for cp: A value-selection heuristic to simulate local search behavior in complete solvers. In *Principles and Practice of Constraint Programming: 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings 24*, pages 99–108. Springer, 2018.

9 Andreas Drexl and Juergen Gruenewald. Nonpreemptive multi-mode resource-constrained project scheduling. *IIE transactions*, 25(5):74–81, 1993.

10 Davide Giglio, Massimo Paolucci, Abdolreza Roshani, and Flavio Tonelli. Multi-manned assembly line balancing problem with skilled workers: a new mathematical formulation. *IFAC-PapersOnLine*, 50(1):1211–1216, 2017.

11 Daniel Godard, Philippe Laborie, and Wim Nuijten. Randomized large neighborhood search for cumulative scheduling. In *ICAPS*, volume 5, pages 81–89, 2005.

12 Shima Javanmard, Behrouz Afshar-Nadjafi, and Seyed Taghi Akhavan Niaki. Preemptive multi-skilled resource investment project scheduling problem: Mathematical modelling and solution approaches. *Computers & Chemical Engineering*, 96:55–68, 2017.

13 Ahmed Karam, El-Awady Attia, and Philippe Duquenne. A milp model for an integrated project scheduling and multi-skilled workforce allocation with flexible working hours. *IFAC-PapersOnLine*, 50(1):13964–13969, 2017.

14 S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. doi:10.1126/science.220.4598.671.

15 RAINER KOLISCH and ANDREAS DREXL. Local search for nonpreemptive multi-mode resource-constrained project scheduling. *IIE Transactions*, 29(11):987–999, 1997. doi:10.1080/07408179708966417.

16 Rainer Kolisch and Sönke Hartmann. Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis. In *Project scheduling*, pages 147–178. Springer, 1999.

17 Stefan Kreter, Andreas Schutt, and Peter J Stuckey. Using constraint programming for solving rcpsp/max-cal. *Constraints*, 22(3):432–462, 2017.

18 Philippe Laborie. An update on the comparison of mip, cp and hybrid approaches for mixed resource allocation and scheduling. In Willem-Jan van Hoeve, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 403–411, Cham, 2018. Springer International Publishing.

19 Haitao Li and Keith Womer. Scheduling projects with multi-skilled personnel by a hybrid milp/cp benders decomposition algorithm. *Journal of Scheduling*, 12(3):281–298, 2009.

20 Paweł B. Myszkowski, Maciej Laszczyk, Ivan Nikulin, and Marek Skowronski. imopse: a library for bicriteria optimization in multi-skill resource-constrained project scheduling problem. *Soft Computing*, 23:3397–3410, 2019.

21 Mireille Palpant, Christian Artigues, and Philippe Michelon. Lssper: Solving the resource-constrained project scheduling problem with large neighbourhood search. *Annals of Operations Research*, 131(1):237–257, 2004.

22 Claude Le Pape and Philippe Baptiste. Heuristic control of a constraint-based algorithm for the preemptive job-shop scheduling problem. *Journal of Heuristics*, 5(3):305–325, 1999.

23 Laurent Perron, Paul Shaw, and Vincent Furnon. Propagation guided large neighborhood search. In *International Conference on Principles and Practice of Constraint Programming*, pages 468–481. Springer, 2004.

24 Oliver Polo-Mejía, Marie-Christine Anselmet, Christian Artigues, and Pierre Lopez. Mixed-integer and constraint programming formulations for a multi-skill project scheduling problem with partial preemption. In *12th International Conference on Modelling, Optimization and Simulation (MOSIM 2018)*, pages 367–374, 2018.

25 Oliver Polo-Mejía, Christian Artigues, Pierre Lopez, and Virginie Basini. Mixed-integer/linear and constraint programming approaches for activity scheduling in a nuclear research facility. *International Journal of Production Research*, 58(23):7149–7166, 2020.

26 Oliver Polo-Mejía, Christian Artigues, Pierre Lopez, Lars Mönch, and Virginie Basini. Heuristic and metaheuristic methods for the multi-skill project scheduling problem with partial preemption. *International Transactions in Operational Research*, 2021.

27 Sacramento Quintanilla, Ángeles Pérez, Pilar Lino, and Vicente Valls. Time and work generalised precedence relationships in project scheduling with pre-emption: An application to the management of service centres. *European Journal of Operational Research*, 219(1):59–72, 2012.

28 Jerzy Rosłon. The multi-mode, resource-constrained project scheduling problem in construction: state of art review and research challenges problem. *Technical Transactions*, 2017. doi:10.4467/2353737XCT.17.070.6427.

29 Mónica A Santos and Anabela P Tereso. On the multi-mode, multi-skill resource constrained project scheduling problem–a software application. In *Soft computing in industrial applications*, pages 239–248. Springer, 2011.

30 Andreas Schutt, Thibaut Feydy, Peter J Stuckey, and Mark G Wallace. Explaining the cumulative propagator. *Constraints*, 16(3):250–282, 2011.

31 Andreas Schutt, Thibaut Feydy, Peter J. Stuckey, and Mark G. Wallace. Solving rcpsp/max by lazy clause generation. *J. Sched.*, 16(3):273–289, 2013. doi:10.1007/s10951-012-0285-x.

32 Andreas Schutt, Thibaut Feydy, Peter J Stuckey, and Mark G Wallace. Solving rcpsp/max by lazy clause generation. *Journal of scheduling*, 16(3):273–289, 2013.

33 Jakob Snauwaert and Mario Vanhoucke. A new algorithm for resource-constrained project scheduling with breadth and depth of skills. *European Journal of Operational Research*, 292(1):43–59, 2021. doi:10.1016/j.ejor.2020.10.032.

34 Jakob Snauwaert and Mario Vanhoucke. A classification and new benchmark instances for the multi-skilled resource-constrained project scheduling problem. *European Journal of Operational Research*, 2022. doi:10.1016/j.ejor.2022.05.049.

35 Tianshu Song, Ke Xu, Jiangneng Li, Yiming Li, and Yongxin Tong. Multi-skill aware task assignment in real-time spatial crowdsourcing. *GeoInformatica*, 24(1):153–173, 2020.

36 Ria Szeredi and Andreas Schutt. Modelling and solving multi-mode resource-constrained project scheduling. In *International Conference on Principles and Practice of Constraint Programming*, pages 483–492. Springer, 2016.

37 Tony Wauters, Joris Kinable, Pieter Smet, Wim Vancroonenburg, Greet Vanden Berghe, and Jannes Verstichel. The multi-mode resource-constrained multi-project scheduling problem. *Journal of Scheduling*, 19(3):271–283, 2016.

38 Kenneth Young, Thibaut Feydy, and Andreas Schutt. Constraint programming applied to the multi-skill project scheduling problem. In *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, pages 308–317, August 2017. doi:10.1007/978-3-319-66158-2_20.

A Appendix

A.1 CP modeling comparisons

We ran new experiments in comparing the two different CP modeling we propose : the default one presented in section 3.2.2 and its upgrade that we called *CP-SmartPreemption*. Computation time are still 1800 s each like in experiments of section 5.1. Results are shown in 4. We interestingly note that *CP-SmartPreemption* is both faster and more efficient than

Table 5 Best solution and time to best solution for CP-Base and *CP-SmartPreemption*.

ID	Features	CP-B	CP-B (s)	CPSmart	CPSmart (s)	ID	Features	CP-B	CP-B (s)	CPSmart	CPSmart (s)
0	A	1080	1.09	1080	0.78	18	B G	3053	0.59	3053	0.20
1	A M	-	-	1080	8.20	19	B G M	3109	81	3109	3.20
2	A G	6534	0.99	6534	0.23	20	B P	2342	133	2340	178
3	A G M	-	-	6534	3.90	21	B M P	-	-	2390	196
4	A P	1104	2.21	1080	64.9	22	B G P	3063	106	3044	30.0
5	A M P	-	-	2495	24.4	23	B G M P	-	-	3180	198
6	A G P	6534	2.40	6534	62.3	24	B C	3196	154	3196	170
7	A G M P	-	-	6612	141	25	B C M	3272	127	3208	88
12	A C P	-	-	-	-	26	B C G	4949	0.52	4949	0.17
13	A C M	-	-	-	-	27	B C G M	4949	2.2	4949	0.80
14	A C G P	-	-	-	-	28	B C P	3252	1.32	3195	217
15	A C G M P	-	-	-	-	29	B C M P	-	-	3208	203
16	B	2339	106	2339	22.9	30	B C G P	4949	1.35	4383	59.0
17	B M	2359	193	2346	158	31	B C G M P	-	-	-	-

the default version. Notably, it manages to find feasible solution to 7 additional instances compared to basic CP formulation. It also succeeds in solving instance ID 30 way more efficiently than CP-Base or even the LNS that was presented in Table 3. As of today, the current modeling of *CP-SmartPreemption* still fails at the most complicated instances on the instance A that have the most operators/workers and number of task.

A.2 Subproblem methods benchmark

We give more hindsight on the general choice done in the paper to use the mixing methods described in section 4.2.2. Naturally we expect that such a portfolio approach that picks a method from a pool of N neighborhood methods can be better in average than using only one method. To confirm this intuition, we run LNS methods using different parameters:

1. "Random selection" methods with parameter f taking values in $[0.1, 0.2, 0.3, 0.4]$
2. "Cut in equal parts" methods with parameter c taking values in $[2, 3, 4, 5, 6]$
3. "Mixing method" : portfolio of the previous 9 methods, chosen randomly at each iteration of LNS.

As a starting experiment, we are testing this methods on 5 classical RCPSP instances with 120 tasks, and run the solver 10 times per neighborhood methods and instances to get a better statistical confidence. We limited to 100 the number of iteration of LNS. Detailed results are in Table 6. It shows empirical confidence in using a mixing method, that achieved best performance. We also ran the same experiment on the instances ID=16,17 in Table 7 and got similar behaviour. Deeper hyper-parameters optimisation could be done for the more complex problems as PP-MS-MM-RCPSP/max-cal in a further research.

Table 6 Neighborhood methods performance on a few instances.

Method	j1201_1	j1201_2	j1201_3	j1201_4	j1201_5
RS(0.1)	116.6	131.3	138.6	106.5	131.3
RS(0.2)	112.2	129.8	136.8	106.0	128.3
RS(0.3)	110.4	128.4	134.3	105.4	120.4
RS(0.4)	108.9	118.0	132.6	103.3	116.7
Cut(2)	**105**	118.0	130	102.0	**113.0**
Cut(3)	107	118.0	135	**101.0**	118.0
Cut(4)	108	126.0	131	106.0	132.0
Cut(5)	111	128.0	136	106.0	130.0
Cut(6)	111	129.0	137	106.0	130.0
Mixing	106.5	**115.0**	**128.3**	102.0	**113.0**

Table 7 Neighborhood methods performance on an industrial use case (ID=16).

Method	ID 16 (B)	ID 17 (BM)
RS(0.1)	2698.0	2745.0
RS(0.2)	2659.0	2675.0
RS(0.3)	2534.0	2635.0
RS(0.4)	2489.0	2662.0
Cut(2)	2344.0	2628.0
Cut(3)	2343.0	**2436.0**
Cut(4)	2346.0	2480.0
Cut(5)	2346.0	2486.0
Cut(6)	2349.0	2776.0
Mixing	**2342**	2460.0

A.3 MSLIB Experiments

To better understand the results we plot the overcost of the LNS method as a function of different parameters used to generate MSLIB4 instances in Figure 4. We theorize that increasing $\#R$ (number of resources) has a negative impact on LNS performance. It also highlights that LNS is underperforming only on instances where $SS = 0$ and $SS\alpha = 0$, defined as skill strength and skill strength variability. As future work we want to continue testing on MSLIB benchmark to possibly improve the LNS method. Detailed results comparing the GA and LNS approach over the tested instances are depicted in Tables 8 and 9.

Table 8 Best results over subset of MSLIB4 instances (404 instance).

Algorithms	LNS (ours)	GA 2021	Equal
#Best Results	90	164	150
Mean Improvement when better solution than GA	2.9%	-	-
Mean Degradation when worse solution than GA	19.2%	-	-

Figure 4 Relative overcost to GA of LNS method, function on different features of MSLIB4 instances.

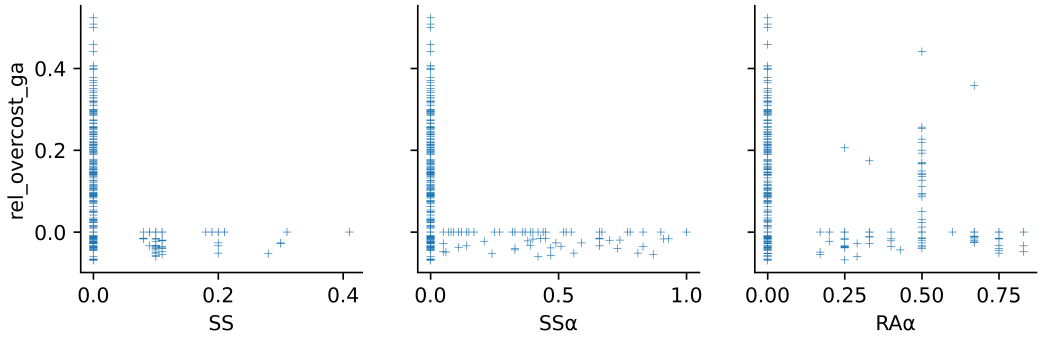


Table 9 Detailed results comparing GA (from [33]) and our methods, function on instance parameters. The GA, Equal, LNS columns counts the number of instance of some given parameters where each of the algorithms got the best results. (LNS-GA)/GA column store the average overcost of LNS method oven the subset of instances described by SP, SS, RA.

SP	SS	RA	NbRun	GA	Equal	LNS	Mean (LNS-GA)/GA (%)
0.1	0.0	0.0	79	41	34	4	12.4
-	-	0.2	40	5	17	18	1.89
-	-	0.3	33	33	0	0	18.6
-	-	0.4	25	9	9	7	5.59
-	-	0.5	66	44	11	11	12.4
-	-	0.6	15	5	7	3	2.86
-	-	0.7	18	11	5	2	9.21
-	-	0.8	22	13	7	2	8.04
-	-	0.9	3	3	0	0	6.78
-	-	1.0	28	0	18	10	-1.01
-	0.1	0.0	24	0	14	10	-1.06
-	-	0.2	6	0	2	4	-2.71
-	-	0.3	3	0	1	2	-3.60
-	-	0.4	1	0	0	1	-4.88
-	-	0.5	1	0	0	1	-1.67
-	-	0.6	1	0	1	0	0.0
-	-	0.7	2	0	1	1	-0.76
-	0.2	0.0	18	0	12	6	-1.00
-	-	0.2	6	0	2	4	-2.67
-	-	0.3	1	0	1	0	0.0
-	-	0.4	2	0	1	1	-2.38
-	0.3	0.0	5	0	3	2	-1.61
-	-	0.2	1	0	0	1	-2.63
-	0.4	0.0	2	0	2	0	0.0
-	0.5	-	2	0	2	0	0.0