New exact and heuristic algorithms to solve the prize-collecting job sequencing problem with one common and multiple secondary resources

Aurélien Froger, Ruslan Sadykov

New exact and heuristic algorithms to solve the prize-collecting job sequencing problem with one common and multiple secondary resources

Aurélien Froger[1,2] Ruslan Sadykov[2]

[1] Université de Bordeaux, CNRS, Bordeaux INP, IMB, UMR 5251, 33405 Talence, France
[2] Centre Inria de l'université de Bordeaux, 33405 Talence, France

Abstract

We study the prize-collecting job sequencing problem with one common and multiple secondary resources. In this problem, a set of jobs is given, each with a profit, multiple time windows for its execution, and a duration during which it requires the main resource. Each job also requires a preassigned secondary resource before, during, and after its use of the main resource. The goal is to select and schedule the subset of jobs that maximize the total profit. We present a new mixed integer linear programming formulation of the problem and a branch-cut-and-price algorithm as an exact solution method. We also introduce a heuristic algorithm to tackle larger instances. Extensive numerical experiments show that our exact algorithm can solve to optimality literature instances with up to 500 jobs for a particular dataset and up to 250 jobs for another dataset with different characteristics. Our heuristic builds high-quality solutions in a small computational time. It computes new best-known solutions for most of the larger instances.

Keywords— scheduling; sequencing; labeling algorithm; branch-cut-and-price; iterated local search

1 Introduction

This work focuses on solving the prize-collecting job sequencing problem with one common and multiple secondary resources (PC-JSOCMSR). This problem was introduced by Horn et al. (2018). It has application in particle therapy scheduling (Maschler and Raidl, 2020) and in pre-runtime scheduling of avionic systems. We refer to the work Horn et al. (2021a) for details about these real-world applications (see §3.3 of the previously cited work).

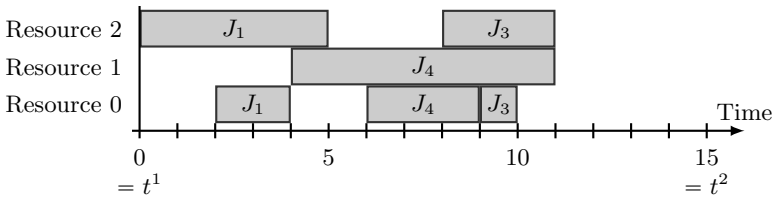
The PC-JSOCMSR is defined as follows. We are given a set $\mathcal{J} = [\![1, n]\!]$ of non-preemptive jobs, meaning that the execution of a job cannot be interrupted and resumed later (hereafter, $[\![a_1, a_2]\!] = [a_1, a_2] \cap \mathbb{Z}$ for every $a_1, a_2 \in \mathbb{Z}$ such that $a_1 \leqslant a_2$). Each job $j \in \mathcal{J}$ should be entirely scheduled (from its start to its end) within one of the time windows of set $\mathcal{W}_j = \{T_{jk} | k \in [\![1, w_j]\!]\}$ with $T_{jk} = [t^1_{jk}, t^2_{jk}]$. Time windows are indexed in chronological order of the t^1_{jk} values. Denoting $t^1 = \min_{j \in \mathcal{J}} \{t^1_{j1}\}$ and $t^2 = \max_{j \in \mathcal{J}} \{t^2_{jw_j}\}$, the planning horizon is defined as the interval $[t^1, t^2]$.

We are also given a set of (renewable) disjunctive resources $\mathcal{R}_0 = \{0\} \cup \mathcal{R}$. We refer to resource 0 as the main resource and to $\mathcal{R} = [\![1, m]\!]$ as the set of secondary resources. Each job $j \in \mathcal{J}$ requires a single secondary resource $r_j \in \mathcal{R}$ (preassigned to j) for p_j consecutive time periods. If the execution of job j starts at time

Email addresses: aurelien.froger@math.u-bordeaux.fr (Aurélien Froger), ruslan.sadykov@inria.fr (Ruslan Sadykov)

Job j	r_j	b_j	p_j			W_j			
			p_j^-	p_j^0	p_j^+	t_{j1}^1	t_{j1}^2	t_{j2}^1	t_{j2}^2
J_1	2	4	2	2	1	0	9	10	15
J_2	1	1	3	3	2	3	12	-	-
J_3	2	2	1	1	1	1	12	-	-
J_4	1	2	2	3	2	4	14	-	-

(a) Instance



(b) Solution

Figure 1: An example of an instance and a feasible solution

period t, then this job requires the main resource (in addition to the resource r_j) from time period $t + p_j^-$ for p_j^0 consecutive time periods (with $p_j^- + p_j^0 \leqslant p_j$). We denote $\hat{p}_r^- = \max_{j \in \mathcal{J}: r_j = r}\{p_j^-\}$ and $\hat{p}_r^+ = \max_{j \in \mathcal{J}: r_j = r}\{p_j^+\}$ where $p_j^+ = p_j - (p_j^- + p_j^0)$. We have $p_j^- \geqslant 0$ and $p_j^+ \geqslant 0$ for every job j. The order of processing starting times on every secondary resource is a suborder of the order of processing starting times on the main resource.

Scheduling a job $j \in \mathcal{J}$ generates a profit $b_j \geqslant 0$, whereas not scheduling it generates no profit. The problem is to schedule the jobs while maximizing the total profit. A feasible solution to the problem is a schedule that satisfies the following constraints: 1) each job is scheduled at most once, 2) if a job is scheduled, it is scheduled uninterruptedly and entirely (from its start to its end) within one of its time windows, and 3) at any time, every resource (main or secondary) is used by at most one job. Hereafter, we refer to constraints 1), 2), and 3) respectively as *elementarity*, *time window*, and *resource usage* constraints. Figure 1 displays a feasible solution to an instance with 4 jobs J_1, J_2, J_3, and J_4. On the right, the vertical axis lists the resources and the horizontal axis represents time. Each rectangle represents the use of a particular resource by a job during a time interval. In the displayed solution, job J_2 is not processed. The starting time of jobs J_1, J_3, and J_4 (i.e., the time at which the preassigned secondary resource starts being used) is equal to 0, 8, and 4 respectively. As $p_{J_4}^- = 2$, job J_4 starts using the main resource at time $6 (= 4 + 2)$ for $p_{J_4}^0 = 3$ time periods. The same is true for the other jobs. The solution value is equal to $b_{J_1} + b_{J_3} + b_{J_4} = 8$.

We now outline the main original contributions of this paper:

- We introduce dominance rules that define properties satisfied by at least one optimal schedule.
- We present a pseudo-polynomial time-indexed mixed integer linear programming (MILP) model for the PC-JSOCMSR. Although indexing variables on time may limit a priori its interest, its numerical performance on existing benchmark instances is largely better than an existing compact order-based MILP model introduced by Horn et al. (2021b), and a significant number of instances are solved to proven optimality in less than two hours.
- We formulate the PC-JSOCMSR as a resource constrained shortest path problem (RCSPP) and describe labeling algorithms to solve it. We discuss different modeling approaches. We present how we use the dominance rules.
- We present a branch-cut-and-price (BCP) algorithm to solve an extended MILP model we introduce for the PC-JSOCMSR. The elementarity constraints are handled in the master problem, which allows to efficiently solve the pricing problem as a RCSPP. Our BCP includes many ingredients found in state-of-the-art BCP algorithms: limited arc memory rank-1 cuts, automatic dual price smoothing stabilization, a procedure for enumerating elementary paths, and a multiphase strong branching procedure.
- We present a heuristic algorithm to initialize the BCP algorithm and to compute high-quality solutions for large-sized instances that remain beyond the reach of an exact algorithm.

Extensive computational experiments on benchmark instances from the literature show that the new exact approaches we present to tackle the PC-JSOCMSR solve to proven optimality instances with up to 250 jobs and

up to 500 jobs for some specific instances. They also show that our heuristic algorithm computes high-quality solutions within short running time and outperforms existing heuristics.

The paper is organized as follows. Section 2 presents a brief overview of related literature. Section 3 presents two dominance rules. Section 4 presents two MILP models for the PC-JSOCMSR, one of which is new. Section 5 presents how we model the problem as a RCSPP. Section 6 describes the BCP algorithm. Section 7 presents the heuristic algorithm. Section 8 shows the results of our computational experiments. Section 9 contains final remarks.

2 Related literature

The PC-JSOCMSR is NP-hard (Horn et al., 2018). It is a recently introduced problem that has been little studied. Particular attention has been paid to the use of multivalued decision diagrams (MDDs). In this context, an MDD is a directed weighted acyclic multigraph (usually layered) storing solutions of the PC-JSOCMSR and their values as paths between a starting node and an ending node. An MDD is said relaxed (resp. restricted) when it encodes a superset (resp. subset) of all feasible solutions. In the top-down compilation procedure, an MDD is built using a dynamic programming formulation of the PC-JSOCMSR with nodes to merge identified using an attached dynamic programming state. If additional nodes are merged to limit the size of the graph, the procedure builds a relaxed MDD. Removing some nodes during the procedure leads to a restricted MDD. In the incremental refinement procedure, a relaxed MDD is first built with one node per layer and is then incrementally refined by removing arcs and splitting nodes to eliminate infeasible solutions. We refer to (Castro et al., 2022) for a survey on the use of MDDs in discrete optimization. Maschler and Raidl (2018) presented top-down compilation (TDC) and incremental refinement procedures to build relaxed multivalued decision diagrams (MDDs) used to obtain dual bounds. To compute heuristic solutions, they built a restricted MDD with a TDC procedure and introduced a general variable neighborhood search (GVNS). The former method dominated the latter in providing quality solutions for instances with less than 100 jobs, while the opposite was more often true for instances with 100 to 300 jobs. To our knowledge, the state-of-the art algorithm for the PC-JSOCMSR is presented in Horn et al. (2021a). It relies on relaxed and restricted MDDs build with a TDC procedure that uses a A* search principle and cost-based filtering procedures to remove arcs in the MDDs. Some instances with 50 jobs are solved to optimality within the 15 minutes time limit. For larger instances, the gap between the value of the best computed solution and the best dual bound is on average between 8 and 15%.

As mentioned by previous works on the PC-JSOCMSR, a related problem is the single-machine scheduling problem introduced by van der Veen et al. (1998), in which a change-over time between the processing of two consecutive tasks must be respected. The value of this change-over time depends on whether the two jobs belongs to the same group. Contrary to this problem where the change-over time is only related to the previously scheduled job, the change-over time incurred before starting a job j in the PC-JSOCMSR depends on two jobs (when different): the last job scheduled on the main resource and the last job scheduled on the secondary resource r_j. There exist additional differences with the PC-JSOCMSR: no time windows and an objective function focused on minimizing the total sequencing time.

More generally, the PC-JSOCMSR shares similarities with single-machine scheduling problems with sequence-dependent setup times. Indeed, a solution to the PC-JSOCMSR can be represented as a sequence of jobs using one after another the main resource. As pointed out before, the setup time prior to running a job is dependent not only on the last scheduled job on the main resource, but also on another job scheduled before. To a certain extent, the problem can be classified as a scheduling problem with past-sequence-dependent setup times, concept introduced by Koulamas and Kyparisis (2008). To our knowledge, the literature on this problem is nonetheless essentially focused on potential learning or deterioration effects related to the past jobs. We refer readers to the survey of Allahverdi (2015). Moreover, another major difference with such scheduling problems is the presence of several time windows in the PC-JSOCMSR.

As mentioned by Horn et al. (2021a), the PC-JSOCMSR is a resource-constrained project scheduling problem (RCPSP) with fixed time lags between some of the tasks and the objective of minimizing a weighted sum of tardy

jobs. To understand this, each job $j \in \mathcal{J}$ is split into three tasks with the adequate precedence constraints. The three tasks require the secondary resource r_j and have duration p_j^-, p_j^0, and p_j^+. The second task also requires the main resource. The weight associated with these tasks are 0 for the first and second task, and b_j for the third one. While making this connection explicit is interesting, it does not seem likely that adapting methods introduced for RCPSPs would be suitable for the PC-JSCOMSR due to the existence of multiple time windows for the tasks in the latter.

Finally, as mentioned by Horn et al. (2021a), the PC-JSOCMSR shares similarities with orienteering problems with time windows. The two problems share the same objective function and time windows should be respected for the start time of the service of a customer in orienteering problems and for the starting time of a job in the PC-JSOCMSR. The two main differences are that there is a travel time between two successive visits to clients in orienteering problems, and that the waiting time before scheduling a task on the main resource in the PC-JSOCMSR depends not only on its time window, but also on the completion time of a job that is not the last one scheduled on the main resource. Despite these differences, taking inspiration from solution methods to orienteering problems with time windows (Vansteenwegen et al., 2011; Gunawan et al., 2016) is of interest for solving the PC-JSOCMSR.

3 Dominance rules

With the aim of reducing the search space, we introduce two dominance rules, directly deduced from the problem definition, that describe properties satisfied by at least one optimal schedule to the PC-JSOCMSR. Specifically, in Proposition 1 and 2 we provide sufficient conditions for restricting the order in which two jobs use one after the other the main resource. These conditions guarantee that a better feasible schedule always exists by reversing the order of the two jobs (Proposition 1), by inserting another job in between or by replacing one of the two jobs (Proposition 2). For each proposition, its precise mathematical description and its proof are given in §S.1 of the supplementary material. Solution methods of the PC-JSOCMSR can take advantage of these results.

Proposition 1. *Let $j, j' \in \mathcal{J}$ such that $j \neq j' \wedge r_j = r_{j'}$, $k \in [\![1, w_j]\!]$, and $k' \in [\![1, w_{j'}]\!]$. The following conditions are sufficient to ensure that there exists an optimal schedule where the main resource is not used by job j' during time window $T_{j'k'}$ and then used consecutively by job j during time window T_{jk}:*

$$t^1_{jk} \leqslant t^1_{j'k'} \ \wedge \ t^2_{jk} \leqslant t^2_{j'k'} \ \wedge \ p_j^- \geqslant p_{j'}^- \ \wedge \ p_j^+ \leqslant p_{j'}^+$$

Proposition 2. *Assume that a job $j' \in \mathcal{J}$ is scheduled during time window $T_{j'k'}$ with $k' \in [\![1, w_{j'}]\!]$. There exists an optimal schedule in which:*

1- *Job j' does not use the main resource just after its use by another job $j'' \in \mathcal{J}\backslash\{j'\}$ during time window $T_{j''k''}$ with $k'' \in [\![1, w_{j''}]\!]$ (resp. j' is not the first job using the main resource) if there exists a job $j \in \mathcal{J}\backslash\{j', j''\}$ (resp. $j \in \mathcal{J}\backslash\{j'\}$) that cannot use the main resource before j'' and after j' (resp. after j') and that satisfies at least one of the two following conditions:*

   a) *Job j uses the same secondary resource as j', is at least as profitable as j', and can always be scheduled in the place of j' regardless of when this latter is scheduled during $T_{j'k'}$.*

   b) *Job j can use the main resource i) just after j'' without preventing j'' to end its execution at time $t^2_{j''k''}$ and imposing restriction on the processing of precedent jobs, and ii) just before j' without preventing j' to start its execution at time $t^1_{j'k'}$ and imposing restriction on the processing of subsequent jobs (resp. ii))*

2- *Job j' does not use the main resource just before its use by another job $j'' \in \mathcal{J}\backslash\{j'\}$ during time window $T_{j''k''}$ with $k'' \in [\![1, w_{j''}]\!]$ (resp. j' is not the last job using the main resource) if there exists a job $j \in \mathcal{J}\backslash\{j', j''\}$ (resp. $j \in \mathcal{J}\backslash\{j'\}$) that cannot use the main resource before j' and after j'' (resp. before j') and that satisfies at least one of the two following conditions:*

a) Job j uses the same secondary resource as j', is at least as profitable as j', and can always be scheduled in the place of j' regardless of when this latter is scheduled during $T_{j'k'}$.

b) Job j can use the main resource i) just after j' without preventing j' to end its execution at time $t^2_{j'k'}$ and imposing restriction on the processing of precedent jobs, and ii) just before j'' without preventing j'' to start its execution at time $t^1_{j''k''}$ and imposing restriction on the processing of subsequent jobs (resp. i))

4 Mixed integer linear programming models

The PC-JSOCMSR can be formulated using mixed linear integer programming. We present two models. The first formulation is a compact MILP model introduced in Horn et al. (2021b). The second formulation is a pseudo-polynomial MILP model that relies on time dicretization.

4.1 Compact order-based MILP model of Horn et al. (2021b)

The compact model introduced by Horn et al. (2021b) is an ordered-based model. It relies on the following decision variables. Binary variable y_j is equal to 1 if job $j \in \mathcal{J}$ is scheduled, 0 otherwise. Binary variable z_{jk} is equal to 1 if job $j \in \mathcal{J}$ is scheduled within its time window T_{jk}. Continuous variable s_j is the starting time period of job j. Binary variable $u_{jj'}$ is equal to 1 if job j finishes using the main resource before its use (not necessarily consecutive) by job j'. Note that when $u_{jj'} = 1$, job j' is not necessarily the next immediate job using the main resource after j. We introduce $\delta_{jj'}$ as the minimum time required between the start of job j and the beginning of another job j' to avoid that these jobs overlap on the main and secondary resources when they are scheduled in this order, but not necessarily consecutively. The value of $\delta_{jj'}$ is equal to p_j if j and j' use both the same secondary resource (i.e., $r_j = r_{j'}$) and to $p^-_j + p^0_j - p^-_{j'}$ if they do not (i.e., $r_j \neq r_{j'}$). The model, denoted $[F1]$, is as follows:

$$[F1] \quad \max \sum_{j \in \mathcal{J}} b_j y_j \tag{1}$$

$$\text{s.t.} \quad \sum_{k=1}^{w_j} z_{jk} = y_j \qquad j \in \mathcal{J} \tag{2}$$

$$u_{jj'} + u_{j'j} \geqslant y_j + y_{j'} - 1 \qquad j, j' \in \mathcal{J}, j \neq j' \tag{3}$$

$$s_{j'} \geqslant s_j + \delta_{jj'} - \big(t^2_{jw_j} - p_j - t^1_{j'1} + \delta_{jj'}\big)(1 - u_{jj'}) \qquad j, j' \in \mathcal{J}, j \neq j' \tag{4}$$

$$s_j \geqslant t^1_{j1} + \sum_{k=1}^{w_j} \big(t^1_{jk} - t^1_{j1}\big) z_{jk} \qquad j \in \mathcal{J} \tag{5}$$

$$s_j \leqslant t^2_{jw_j} - p_j + \sum_{k=1}^{w_j} \big(t^2_{jk} - t^2_{jw_j}\big) z_{jk} \qquad j \in \mathcal{J} \tag{6}$$

$$y_j \in \{0, 1\} \qquad j \in \mathcal{J} \tag{7}$$

$$z_{jk} \in \{0, 1\} \qquad j \in \mathcal{J}, k \in \{1, \ldots, w_j\} \tag{8}$$

$$s_j \in [t^1_{j1}, t^2_{jw_j} - p_j] \qquad j \in \mathcal{J} \tag{9}$$

$$u_{jj'} \in \{0, 1\} \qquad j, j' \in \mathcal{J}, j \neq j' \tag{10}$$

The objective function (1) maximizes the total profit for the scheduled jobs. Constraints (2) state that a job is deemed scheduled if it is scheduled within one of its time window. Constraints (3) impose a linear order for the use of the main resource for every couple of jobs that are scheduled. Constraints (4) ensure that jobs are not using the same resources simultaneously. Specifically, according to the order in which they use the main resource, the constraints guarantee that the starting times of two scheduled jobs is sufficient to avoid overlapping. Constraints (5) and (6) impose that jobs are scheduled within their time windows. Finally, constraints (7)–(10) set the domain of the decision variables.

4.2 Pseudo-polynomial time-indexed MILP model

Model $[F1]$ is a continuous-time model as the start time of every jobs is a continuous variable that can take an infinite number of values provided that the constraints of the problem are respected. We introduce a pseudo-polynomial formulation that relies on time discretization. If the data is not integer, provided that the granularity of the time discretization is fine enough, a discrete-time formulation has the same optimal value as a continuous-time formulation of the same problem. Available benchmark instances introduced in previous studies use integer data (see Section 8).

The model, denoted $[F2]$, uses the following decision variables. As in $[F1]$, binary variable y_j is equal to 1 if job $j \in \mathcal{J}$ is scheduled, 0 otherwise. We denote $\mathcal{T}_j$ as the set of possible starting time periods for job $j \in \mathcal{J}$. Specifically we have $\mathcal{T}_j = \bigcup_{k \in \{1,\dots,w_j\}} \{t^1_{jk}, t^1_{jk}+1, \dots, t^2_{jk}-p_j\}$. We denote as $\mathcal{T}$ the set of time periods (i.e., $\mathcal{T} = \{t^1, t^1+1, \dots, t^2\}$). Binary variable x_{jt} is equal to 1 if job $j \in \mathcal{J}$ starts being executed at time period $t \in \mathcal{T}_j$, 0 otherwise. The model is as follows:

$$[F2] \quad \max \sum_{j \in \mathcal{J}} b_j y_j \tag{11}$$

$$\text{s.t.} \quad \sum_{t \in \mathcal{T}_j} x_{jt} = y_j \qquad j \in \mathcal{J} \tag{12}$$

$$\sum_{j \in \mathcal{J}} \sum_{t' \in T_j \cap \{t-p_j^- - p_j^0 + 1, \dots, t-p_j^-\}} x_{jt'} \leqslant 1 \qquad t \in \mathcal{T} \tag{13}$$

$$\sum_{\substack{j \in \mathcal{J} \\ \wedge r_j = r}} \sum_{t' \in T_j \cap \{t-p_j+1, \dots, t\}} x_{jt'} \leqslant 1 \qquad t \in \mathcal{T}, r \in \mathcal{R} \tag{14}$$

$$y_j \in \{0,1\} \qquad j \in \mathcal{J} \tag{15}$$

$$x_{jt} \in \{0,1\} \qquad j \in \mathcal{J}, t \in \mathcal{T}_j \tag{16}$$

The objective function (11) is identical to the one of model $[F1]$ and maximizes the total profit for the scheduled jobs. Constraints (12) state that a job is deemed scheduled if its start is scheduled at any time period that is adequate with respect to its time windows. Constraints (13) and (14) ensure that jobs are not using the same resources simultaneously. Specifically, the number of jobs using the main resource and each secondary resource at any time $t \in \mathcal{T}$ is less than or equal to one. Constraints (15) and (16) set the domains of the decision variables.

5 Graph-based model

Previous studies on the PC-JSOCMSR using MDDs introduce very large acyclic graphs where every arc represents the processing of a job with a weight equal to its profit and every path between a starting node and an ending node represents a solution to the PC-JSOCMSR (Maschler and Raidl, 2018; Horn et al., 2021a). An optimal solution is then computed by solving a shortest path problem. If the MDD is relaxed, this solution may be infeasible by potentially violating elementarity, time window, and resource usage constraints.

In this section, we reformulate the PC-JSOCMSR as a resource-constrained shortest path problem (RCSPP) on a small graph that contains cycles. Every arc represents the processing of a job or a transition between the use of the main resource by two jobs. It has a cost and a vector of resource consumption. A path between the source and the sink of this graph represents a feasible solution to the PC-JSOCMSR (i.e., a schedule) if and only if at each node of this path the bounds on the resource consumption accumulated along the path are not violated. An optimal schedule for the PC-JSOCMSR can then be obtained by solving the RCSPP on this small graph by the labeling algorithm presented below. We start with the graph model which describes all feasible schedules, before adapting it to the case when only semi-active schedules are admitted.

We formulate the PC-JSOCMSR as an RCSPP on a directed graph $G = (V, A)$ defined from a set V of nodes and a set A of arcs between these nodes. There are three special nodes v_{source}, $v_\star$ and v_{sink} in V that are used to model, respectively, the beginning of a schedule, any intermediate moment between the use of the

main resource by two tasks, and the end of a schedule. The set V also contains a node $v_{j,k}$ for every job j and time window $k \in \{1, \ldots, w_j\}$. We denote by $V_j = \{v_{j,k} : k \in \{1, \ldots, w_j\}\}$ the subset of nodes associated with job j. The set of arcs A contains the arcs $(v_{source}, v_\star)$ and $(v_\star, v_{sink})$, as well as arcs $(v_\star, v_{j,k})$ and $(v_{j,k}, v_\star)$ for every pair (j, k). Traversing the arc $(v_{j,k}, v_\star)$ in a path corresponds to processing job j during its time window T_{jk} in the corresponding schedule. Each path from v_{source} to v_{sink} in G defines the jobs to be scheduled and the order in which these jobs use the main resource. There also exists an idle (loop) arc $(v_\star, v_\star)$ to model idles times that are required due to the presence of time windows and sometimes necessary when the earliest starting time of a job inserted at the end of a partial schedule is different when we calculate it independently for the main resource and its secondary resource. We denote as $\Gamma^-(v)$ and $\Gamma^+(v)$ the set of arcs entering in and leaving every node v, respectively. A path starting at v_{source} is called a forward path, while a path starting at v_{sink} and composed of arcs traversed in their opposite direction is called a backward path. Any forward (resp. backward) path defines a partial schedule that is built from time t^1 (resp. built backward from time t^2).

To ensure elementarity, time window and resource usage constraints, we associate a resource consumption to each arc, these consumed resources being in one-to-one correspondence with the resources of set $\mathcal{R}_0$. The accumulated consumption of resource 0 represents time. The accumulated consumption of resource $r \in \mathcal{R}$ represents the time to be added to the current accumulated consumption of resource 0 to obtain the time when r becomes available for jobs. Let Q_0 be the accumulated consumption of resource 0 (i.e., the current time) and Q_r be the accumulated consumption of a secondary resource r. In a forward (resp. backward) path, this means that the resource 0 is available during the time interval $[Q_0, t^2)$ (resp. $[t^1, Q_0)$), and that resource r is available during the time interval $[Q_0 + Q_r, t^2)$ (resp. $[t^1, Q_0 + Q_r)$). In a forward (resp. backward) path, a negative (resp. positive) value of the accumulated consumption of a secondary resource means that the resource is already available for some time, whereas a positive (resp. negative) value means that the resource is currently used. The consumption $q_{a,r}$ of resource r when traversing arc $a \in A$ is defined as follows:

$$q_{a,r} = \begin{cases} 0 & \text{if } a = (v_{source}, v_\star) \vee a \in \Gamma^+(v_\star) \backslash \{(v_\star, v_\star)\} \\ p_j^0 & \text{if } a = (v_{j,k}, v_\star) \text{ and } r = 0 \\ -p_j^0 & \text{if } a = (v_{j,k}, v_\star) \text{ and } r \in \mathcal{R} \backslash \{r_j\} \\ p_j^- + p_j^+ & \text{if } a = (v_{j,k}, v_\star) \text{ and } r = r_j \\ 1 & \text{if } a = (v_\star, v_\star) \text{ and } r = 0 \\ -1 & \text{if } a = (v_\star, v_\star) \text{ and } r \in \mathcal{R} \end{cases}$$

As a path defines the order in which jobs use the main resource, we increase the accumulated consumption of resource 0 (i.e., we increase the current time) by p_j^0 when scheduling job j, which requires subtracting p_j^0 from the accumulated consumption of the secondary resources different from r_j because the time intervals during which they are continuously available are not modified. If job j starts using the main resource at the current time, it means that it started using r_j for p_j^- time units. After j stops using the main resource, it still uses resource r_j for p_j^+ time units, which explains the arc consumption for this resource. As traversing the idle arc $(v_\star, v_\star)$ means going to next time period, we increase the current time by one, which requires subtracting one for the accumulated consumption of every secondary resource (because the time intervals during which they are continuously available are not modified).

Each node $v \in V$ is given an interval $[l_{v,r}, u_{v,r}]$ for the accumulated consumption of each resource $r \in \mathcal{R}_0$. The intervals are defined as follows:

$$[l_{v,r}, u_{v,r}] = \begin{cases} [t^1, t^1] & \text{if } v = v_{source} \text{ and } r = 0 \\ [0, 0] & \text{if } v = v_{source} \text{ and } r \in \mathcal{R} \\ [t^1, t^2] & \text{if } v = v_\star \text{ and } r = 0 \\ [-\hat{p}_r^-, \hat{p}_r^+] & \text{if } v = v_\star \text{ and } r \in \mathcal{R} \\ [t^2, t^2] & \text{if } v = v_{sink} \text{ and } r = 0 \\ [0, 0] & \text{if } v = v_{sink} \text{ and } r \in \mathcal{R} \\ [t_{jk}^1 + p_j^-, t_{jk}^2 - (p_j^0 + p_j^+)] & \text{if } v = v_{j,k} \text{ and } r = 0 \\ [-p_j^-, -p_j^-] & \text{if } v = v_{j,k} \text{ and } r = r_j \\ [-\hat{p}_r^-, \hat{p}_r^+] & \text{if } v = v_{j,k} \text{ and } r \in \mathcal{R} \backslash \{r_j\} \end{cases}$$

The node v can be traversed by a forward path (resp. by a backward path) only if the resource consumption accumulated up to v belongs to $[l_{v,0}, u_{v,0}]$ for resource 0 and is less than or equal to $u_{v,r}$ (resp. larger than or equal to $l_{v,r}$) for the secondary resources. For a node $v_{j,k}$, the interval $[t^1_{jk} + p^-_j, t^2_{jk} - (p^0_j + p^+_j)]$ for resource 0 guarantees that job j is executed during time window T_{jk}. The interval $[-p^-_j, -p^-_j]$ for secondary resource r_j makes sure that this resource is available for long enough when job j starts using the main resource. Indeed, job j can start using resource 0 in a forward (resp. backward) path at the current time Q_0 only if the resource r_j is available for a time interval that contains $[Q_0 - p^-_j, t^2)$ (resp. $[t^1, Q_0 - p^-_j)$). The intervals $[-\hat{p}^-_r, \hat{p}^+_r]$ for the secondary resources different from r_j are not restrictive in the sense they do not impact the respect of past and future accumulated consumption intervals. Specifically, in a forward (resp. backward) path, every secondary resource r is available in time interval $[Q_0 + \hat{p}^+_r, t^2)$ (resp. $[t^1, Q_0 - \hat{p}^-_r)$) if the current time is Q_0.

The intervals for the accumulated consumption of secondary resources also have the following characteristic which is essential for building a feasible schedule. An accumulated consumption in a forward (resp. backward) path that is smaller than $l_{v,r}$ (resp. larger than $u_{v,r}$) should be updated to $l_{v,r}$ (resp. $u_{v,r}$). This is necessary in the following cases. Let Q_0 be the accumulated consumption of resource 0 (i.e., the current time) and Q_r be the accumulated consumption of every secondary resource $r \in \mathcal{R}$. If $Q_0 + Q_{r_j} < Q_0 - p^-_j$ in a forward path, then job j can start using resource 0 at time Q_0, which means it starts using resource r_j at time $Q_0 - p^-_j$. An increase of Q_{r_j} to $-p^-_j$ is therefore necessary to account for the starting time of job j and to ensure that the accumulated consumption of resource r_j becomes equal to p^+_j after traversing the arc $(v_{j,k}, v_\star)$. Similarly, if $Q_0 + Q_{r_j} > Q_0 - p^-_j$ in a backward path, then job j can start using resource 0 at time Q_0 (time is considered in backward direction), which means that j starts using resource r_j at time $Q_0 - p^-_j$. A decrease of Q_{r_j} to $-p^-_j$ is therefore necessary to account for the fact that r_j is then available only during the time interval $[t^1, Q_0 - p^-_j)$. If, in a forward (resp. backward) path, a resource r different from r_j is available at time Q_0 during a time interval that contains $[Q_0 - \hat{p}^-_r, t^2)$ (resp. $[t_1, Q_0 + \hat{p}^+_r)$), meaning that $Q_r < -\hat{p}^-_r$ (resp. $Q_r > \hat{p}^+_r$), then increasing (resp. decreasing) Q_r to $-\hat{p}^-_r$ (resp. $\hat{p}^+_r$) is valid because no job that could start using resource 0 at time Q_0 would require the use of r before time $Q_0 - \hat{p}^-_r$ (resp. after time $Q_0 + \hat{p}^+_r$).

We finally define the cost c_a of each arc $a \in A$ as:

$$c_a = \begin{cases} b_j & \text{if } a \in \Gamma^+(v_{j,k}) \\ 0 & \text{otherwise} \end{cases}$$

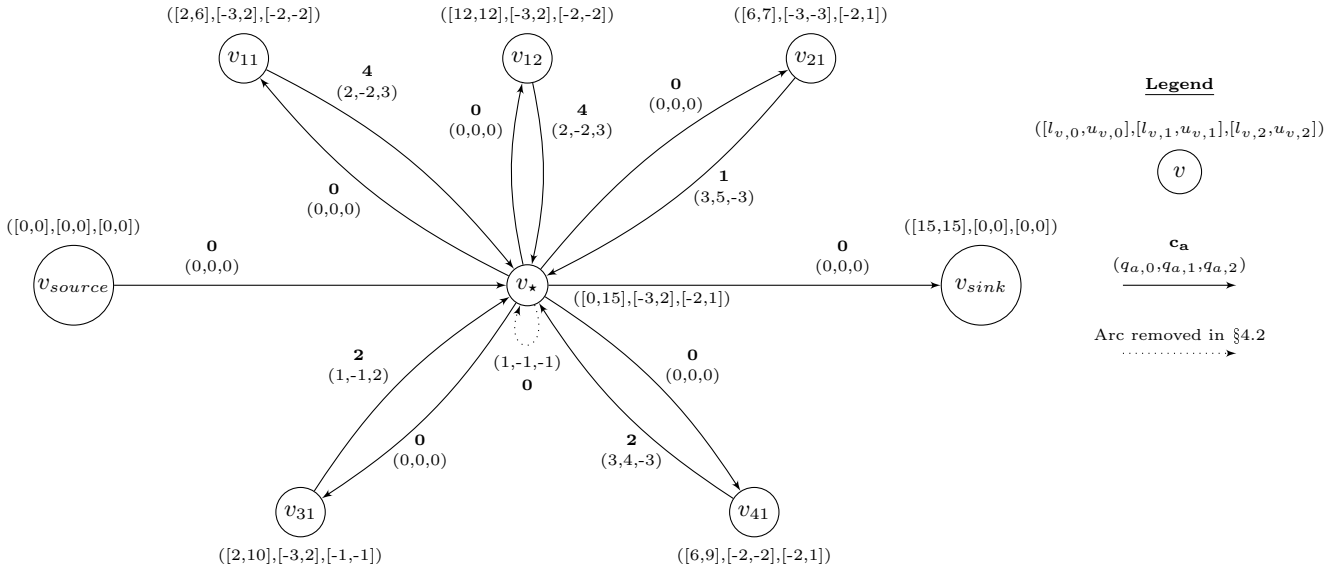


Figure 2: Graph G associated with the instance of Figure 1

To illustrate the graph formulation of the problem, we show in Figure 2 the graph G associated with the instance of Figure 1. The solution of Figure 1 is represented in this graph by the following path: $(v_{source}, v_\star)$, $(v_\star, v_\star)$, $(v_\star, v_\star)$, $(v_\star, v_{11})$, $(v_{11}, v_\star)$, $(v_\star, v_\star)$, $(v_\star, v_\star)$, $(v_\star, v_{41})$, $(v_{41}, v_\star)$, $(v_\star, v_{31})$, $(v_{31}, v_\star)$, $(v_\star, v_\star)$, $(v_\star, v_\star)$, $(v_\star, v_\star)$, $(v_\star, v_\star)$, $(v_\star, v_\star)$, $(v_\star, v_{sink})$. This path can be computed by the labeling algorithms introduced in §5.1.

The PC-JSOCMSR is then equivalent to an RCSPP whose goal is to find the path from v_{source} and v_{sink} in G with maximum cost that satisfies all the resource consumption interval and elementarity constraints. The RCSPP with elementarity constraints is NP-hard for the case of a general graph with cycles (Dror, 1994). We describe in §5.1 the labeling algorithms used to solve this RCSPP. We present in §5.2 how we can modify the label extension to consider only semi-active schedules. We then present in §5.3 an alternative definition of graph G that allows us to take into account the dominance rules introduced in Section 3.

5.1 Forward and backward labeling algorithms

Enumeration of all feasible (resource-constrained) paths in graph G can be performed by a labeling algorithm applying bi-directional search (i.e., a forward search from v_{source} and a backward search from v_{sink}).

Forward labeling algorithm The forward labeling algorithm uses a forward label $\vec{L} = \left(c^{\vec{L}}, v^{\vec{L}}, Q^{\vec{L}}, J^{\vec{L}}\right)$ to represent a path starting at v_{source} and finishing at node $v^{\vec{L}} \in V$ with accumulated resource consumption $Q^{\vec{L}}$ and a cost equal to $c^{\vec{L}}$. Forward label $\vec{L}$ is also characterized by a set $J^{\vec{L}}$ of scheduled jobs. The algorithm starts with one unextended label $(0, v_{source}, (l_{v_{source},r})_r, \varnothing)$. At each iteration, it selects an unextended label $\vec{L}$ such that $v^{\vec{L}} \neq v_{sink}$, proceeds to its extension along every arc $a = (v^{\vec{L}}, v) \in \Gamma^+(v^{\vec{L}})$ such that $v \notin \bigcup_{j \in J^{\vec{L}}} V_j$, and marks $\vec{L}$ as extended. When extending the label $\vec{L}$ along an arc a, it creates an unextended label $\vec{L'}$ with the following characteristics:

$$
\begin{aligned}
c^{\vec{L'}} &= c^{\vec{L}} + c_a \\
v^{\vec{L'}} &= v \\
Q_r^{\vec{L'}} &= \begin{cases} Q_r^{\vec{L}} + q_{a,r} & r = 0 \\ \max\left\{l_{v,r}\ ,\ Q_r^{\vec{L}} + q_{a,r}\right\} & r \in \mathcal{R} \end{cases} \\
J^{\vec{L'}} &= \begin{cases} J^{\vec{L}} \cup \{j\} & v \in V_j \\ J^{\vec{L}} & \text{otherwise} \end{cases}
\end{aligned}
$$

The computation of $Q_r^{\vec{L'}}$ for $r \in \mathcal{R}$ can be interpreted as follows: if the resource r is available at the current time $Q_0^{\vec{L'}}$ for more than $l_{v,r}$ time periods (meaning that the $Q_r^{\vec{L}} + q_{a,r} < l_{v,r}$), then $Q_r^{\vec{L'}}$ can always be at least set equal to $l_{v,r}$ as no job that would start using resource 0 at $Q_0^{\vec{L'}}$ could use r before time $Q_0^{\vec{L'}} - l_{v,r}$ (according to the values set for $l_{v,r}$). The label $\vec{L'}$ is deleted if $Q_0^{\vec{L'}} \notin [l_{v,0}, u_{v,0}]$ or there exists $r \in \mathcal{R}$ such that $Q_r^{\vec{L'}} > u_{v,r}$. The algorithm continues as long as there are still unextended labels at a node $v \in V \backslash \{v_{sink}\}$. After the algorithm termination, the set of labels at v_{sink} corresponds to the set of feasible resource-constrained paths in G from v_{source} to v_{sink}.

Backward labeling algorithm The backward labeling algorithm uses a backward label $\overleftarrow{L} = (\bar{c}^{\overleftarrow{L}}, v^{\overleftarrow{L}}, Q^{\overleftarrow{L}}, J^{\overleftarrow{L}})$ to represent a (backward) path starting at v_{sink} and finishing at node $v^{\overleftarrow{L}} \in V$. It starts with one unextended label $(0, v_{sink}, (u_{v_{sink},r})_r, \varnothing)$. At each iteration, it selects an unextended label $\overleftarrow{L}$ such that $v^{\overleftarrow{L}} \neq v_{source}$ and extend it along every arc $a = (v, v^{\overleftarrow{L}}) \in \Gamma^-(v^{\overleftarrow{L}})$ (considering a in its opposite direction) such that $v \notin \bigcup_{j \in J^{\overleftarrow{L}}} V_j$, and marks $\overleftarrow{L}$ as extended. When extending the label $\overleftarrow{L}$ along an arc a, we create an unextended label $\overleftarrow{L'}$ with the following characteristics:

$$
\begin{aligned}
c^{\overleftarrow{L'}} &= c^{\overleftarrow{L}} + c_a \\
v^{\overleftarrow{L'}} &= v
\end{aligned}
$$

$$Q_r^{\overleftarrow{L}'} = \begin{cases} Q_r^{\overleftarrow{L}} - q_{a,r} & r = 0 \\ \min\left\{u_{v,r}\ , Q_r^{\overleftarrow{L}} - q_{a,r}\right\} & r \in \mathcal{R} \end{cases}$$

$$J^{\overleftarrow{L}'} = \begin{cases} J^{\overleftarrow{L}} \cup \{j\} & v \in V_j \\ J^{\overleftarrow{L}} & \text{otherwise} \end{cases}$$

The label $\overleftarrow{L}'$ is deleted if $Q^{\overleftarrow{L}'} \notin [l_{v,0}, u_{v,0}]$ or if there exists $r \in \mathcal{R}$ such that $Q_r^{\overleftarrow{L}'} < l_{v,r}$. The algorithm continues as long as there are still unextended labels at a node $v \in V \backslash \{v_{source}\}$.

In the forward and backward labeling algorithms, we extend a forward label $\vec{L}$ (resp. backward label $\overleftarrow{L}$) in non-decreasing (resp. non-increasing) order of the vector $Q_r^{\vec{L}}$ (resp. $Q_r^{\overleftarrow{L}}$), with two labels being compared using the lexicographical order on the Cartesian product $\mathbb{Z}^{m+1}$. Since G only contains cycles where the consumption of resource 0 increases (resp. decreases) by at least 1, these algorithms are label-setting algorithms, meaning that they only extend labels that can never be dominated by another label created after that extension. The maximum number of labels in these algorithms is bounded above by $1 + 2^n\Big((t^2 - t^1)\prod_{r\in\mathcal{R}}(\hat{p}_r^+ - \hat{p}_r^-) + \sum_{j\in\mathcal{J}}\sum_{k=1}^{w_j}\left((t_{jk}^2 - t_{jk}^1 - p_j)\prod_{r\in\mathcal{R}\backslash\{r_j\}}(\hat{p}_r^+ - \hat{p}_r^-)\right)\Big)$. In the worst case, each label is extended for every possible addition of a job to the partial schedule The time and space complexity of the labeling algorithms are therefore exponential in the worst case.

Bi-directional labeling algorithm We run the forward and backward algorithms where only labels $\vec{L}$ such that $Q_0^{\vec{L}} \leqslant t^*$ and labels $\overleftarrow{L}$ such that $Q_0^{\overleftarrow{L}} > t^*$ are kept *(an extension that leads to a label that does not have this property is deleted)*. The value of t^* is set equal to $\frac{t_1+t_2}{2}$. We then compute all the paths from v_{source} to v_{sink} by concatenating a forward path, an arc, and a backward path. We can concatenate a (partial) path represented by a forward label $\vec{L}$ and a (partial) backward path represented by a backward label $\overleftarrow{L}$ along an arc $a = (v^{\vec{L}}, v^{\overleftarrow{L}}) \in A$ if the following conditions are satisfied:

$$\begin{aligned} & Q_0^{\vec{L}} + q_{a,0} = Q_0^{\overleftarrow{L}} \\ & Q_r^{\vec{L}} + q_{a,r} \leqslant Q_r^{\overleftarrow{L}} \quad r \in \mathcal{R} \\ & J^{\vec{L}} \cap J^{\overleftarrow{L}} = \varnothing \end{aligned}$$

The cost of the resulting path is equal to $c^{\vec{L}} + c_a + c^{\overleftarrow{L}}$.

The forward (resp. backward) labels computed when traversing forward (resp. backward) the path associated with the solution of Figure 1 are as follows:

(Forward) $(0, v_{source}, 0, 0, 0, \{\}) \to (0, v_\star, 0, 0, 0, \{\}) \to (0, v_\star, 1, -1, -1, \{\}) \to (0, v_\star, 2, -2, -2, \{\}) \to (0, v_{11}, 2, -2, -2, \{J_1\}) \to$
$(4, v_\star, 4, -3, 1, \{J_1\}) \to (4, v_\star, 5, -3, 0, \{J_1\}) \to (4, v_\star, 6, -3, -1, \{J_1\}) \to (4, v_{41}, 6, -2, -1, \{J_1\}) \to (6, v_\star, 9, 2, -2, \{J_1, J_4\}) \to$
$(6, v_{31}, 9, 2, -1, \{J_1, J_4\}) \to (8, v_\star, 10, 1, 1, \{J_1, J_3, J_4\}) \to (8, v_\star, 11, 0, 0, \{J_1, J_3, J_4\}) \to (8, v_\star, 12, -1, -1, \{J_1, J_3, J_4\}) \to$
$(8, v_\star, 13, -2, -2, \{J_1, J_3, J_4\}) \to (8, v_\star, 14, -3, -2, \{J_1, J_3, J_4\}) \to (8, v_\star, 15, -3, -2, \{J_1, J_3, J_4\}) \to (8, v_{sink}, 15, 0, 0, \{J_1, J_3, J_4\})$

(Backward) $(0, v_{sink}, 15, 0, 0, \{\}) \to (0, v_\star, 15, 0, 0, \{\}) \to (0, v_\star, 14, 1, 1, \{\}) \to (0, v_\star, 13, 2, 1, \{\}) \to (0, v_\star, 12, 2, 1, \{\}) \to (0, v_\star, 11, 2, 1, \{\}) \to$
$(0, v_\star, 10, 2, 1, \{\}) \to (2, v_{31}, 9, 2, -1, \{J_3\}) \to (2, v_\star, 9, 2, -1, \{J_3\}) \to (4, v_{41}, 6, -2, 1, \{J_3, J_4\}) \to (4, v_\star, 6, -2, 1, \{J_3, J_4\}) \to$
$(4, v_\star, 5, -1, 1, \{J_3, J_4\}) \to (4, v_\star, 4, 0, 1, \{J_3, J_4\}) \to (8, v_{11}, 2, 2, -2, \{J_1, J_3, J_4\}) \to (8, v_\star, 1, 2, -1, \{J_1, J_3, J_4\}) \to$
$(8, v_\star, 0, 2, 0, \{J_1, J_3, J_4\}) \to (8, v_{source}, 0, 0, 0, \{J_1, J_3, J_4\})$

5.2 Semi-active schedules, modification of label extension and dominance checks

The set of semi-active schedules (i.e., schedules that do not admit local left shifts for every job) is dominant for the PC-JSOCMSR, as any schedule can be transformed into a semi-active one. We can therefore restrict the search of an optimal solution to the PC-JSOCMSR to these type of schedules. Due to the idle arc $(v_\star, v_\star)$, schedules associated to feasible solutions of the RCSPP are not necessary semi-active. To limit the enumeration to semi-active schedules, we remove this idle arc from A and adapt the RCSPP definition. The resource consumption accumulated in a forward (resp. backward) path up to v should be less than or equal to $u_{v,0}$ (resp. larger than or equal to $l_{v,0}$). The update of the accumulated consumption of the secondary resources according

to the intervals defined at each node is done in a similar way as described before, but an additional update is introduced to handle the insertion of idle times. Let Q_0 be the accumulated consumption of resource 0 (i.e., the current time) and Q_r be the accumulated consumption of a secondary resource r. In a forward path up to a node v, if Q_0 is less than $l_{v,0}$ (the job cannot start using resource 0), then the time should increase at least by $l_{v,0} - Q_0$. Similarly, if $Q_r \geqslant u_{v,r}$ (resource r is not available for long enough at the current time), then the time should increase at least by $Q_r - u_{v,r}$ as this results in subtracting the opposite of this value to Q_r. The path corresponds to a feasible schedule if the maximum of these minimum time increases (corresponding to a number of idle times) is such that the resulting time is less than or equal to $u_{v,0}$. Similarly, in a backward path up to a node v, if Q_0 is larger than $u_{v,0}$, then the time should decrease at least by $Q_0 - u_{v,0}$. If $Q_r \leqslant l_{v,r}$, then the time should decrease at least by $l_{v,r} - Q_r$. The backward path corresponds to a feasible schedule if the maximum of these minimum time decreases is such that the resulting time is larger than or equal to $l_{v,0}$. Unlike forward paths that are associated with semi-active schedules, a backward path is associated with a schedule that does not admit local right shifts for every job. Further in the text, when concatenating a forward path with a backward path, we derive a semi-active schedule by local left shifting of the jobs of the latter path.

We describe below the modifications that are required in the forward, backward, and bi-directional algorithms.The time and space complexity of these algorithms in the worst case remains exponential. Pseudo-code of these algorithms are available in §S.2 of the supplementary material.

- **Forward labeling algorithm**
  The value of the accumulated resource consumption of a label $\vec{L'}$, result of the extension of $\vec{L}$ along an arc $a = (v^{\vec{L}}, v)$, becomes:

$$Q_r^{\vec{L'}} = \begin{cases} Q_r^{\vec{L}} + q_{a,r} + \vec{\Delta}(\vec{L}, a) & r = 0 \\ \max\left\{l_{a,r}\ ,\ Q_r^{\vec{L}} + q_{a,r} - \vec{\Delta}(\vec{L}, a)\right\} & r \in \mathcal{R} \end{cases}$$

  where

$$\vec{\Delta}(\vec{L}, a) = \max\left\{0,\ l_{v,0} - Q_0^{\vec{L}} - q_{a,0}\ ,\ \max_{r\in\mathcal{R}}\left\{Q_r^{\vec{L}} + q_{a,r} - u_{v,r}\right\}\right\}$$

  The value of $\vec{\Delta}(\vec{L}, a)$ corresponds to the minimum idle time before the job associated with node v starts using the main resource. The value $l_{v,0} - Q_0^{\vec{L}} - q_{a,0}$ corresponds to the idle time before the beginning of the time window of this job, while the value $\max_{r\in\mathcal{R}}\left\{Q_r^{\vec{L}} + q_{a,r} - u_{v,r}\right\}$ corresponds to the idle time necessary to ensure that the secondary resources are available for long enough. The value of $Q_0^{\vec{L'}}$ ultimately represents the time at which the job (possibly dummy) associated with node v starts using the main resource. The label $\vec{L'}$ is deleted if $Q_0^{\vec{L'}} > u_{v,0}$.

- **Backward labeling algorithm**
  The value of accumulated resource consumption of a label $\overleftarrow{L'}$, result of the extension of $\overleftarrow{L}$ along an arc $a = (v, v^{\overleftarrow{L}})$ (traversed backward), becomes:

$$Q_r^{\overleftarrow{L'}} = \begin{cases} Q_r^{\overleftarrow{L}} - q_{a,r} - \overleftarrow{\Delta}(\overleftarrow{L}, a) & r = 0 \\ \min\left\{u_{v,r}\ ,\ Q_r^{\overleftarrow{L}} - q_{a,r} + \overleftarrow{\Delta}(\overleftarrow{L}, a)\right\} & r \in \mathcal{R} \end{cases}$$

  where

$$\overleftarrow{\Delta}(\overleftarrow{L}, a) = \max\left\{0,\ Q_0^{\overleftarrow{L}} - q_{a,0} - u_{v,0}\ ,\ \max_{r\in\mathcal{R}}\left\{l_{v,r} - (Q_r^{\overleftarrow{L}} - q_{a,r})\right\}\right\}$$

  The value of $\overleftarrow{\Delta}(\overleftarrow{L}, a)$ represents a number of idle time periods for the main resource. The label $\overleftarrow{L'}$ is deleted if $Q_0^{\overleftarrow{L'}} < l_{v,0}$.

- **Bi-directional labeling algorithm**
  The two first conditions for concatenating (a path represented by) a forward label $\vec{L}$ and (a path represented by) a backward label $\overleftarrow{L}$ along an arc $a = (v^{\vec{L}}, v^{\overleftarrow{L}})$ become:

$$Q_0^{\vec{L}} + q_{a,0} \leqslant Q_0^{\overleftarrow{L}}$$
$$Q_r^{\vec{L}} + q_{a,r} - \left(Q_0^{\overleftarrow{L}} - (Q_0^{\vec{L}} + q_{a,0})\right) \leqslant Q_r^{\overleftarrow{L}} \quad r \in \mathcal{R}$$

The solution of Figure 1 is a semi-active schedule and can therefore be represented (due to the modification of the labeling algorithms) as the following path in the graph G without idle arc: $(v_{source}, v_\star)$, $(v_\star, v_{11})$, $(v_{11}, v_\star)$, $(v_\star, v_{41})$, $(v_{41}, v_\star)$, $(v_\star, v_{31})$, $(v_{31}, v_\star)$, $(v_\star, v_{sink})$. The forward (resp. backward) labels computed when traversing the path forward (resp. backward) are as follows:

(Forward) $(0, v_{source}, 0, 0, 0, \{\}) \rightarrow (0, v_\star, 0, 0, 0, \{\}) \rightarrow (0, v_{11}, 2, -2, -2, \{\}) \rightarrow (4, v_\star, 4, -3, 1, \{J_1\}) \rightarrow (4, v_{41}, 6, -2, -1, \{J_1\}) \rightarrow$
$(6, v_\star, 9, 2, -2, \{J_1, J_4\}) \rightarrow (6, v_{31}, 9, 2, -1, \{J_1, J_4\}) \rightarrow (8, v_\star, 10, 1, 1, \{J_1, J_3, J_4\}) \rightarrow (8, v_{sink}, 15, 0, 0, \{J_1, J_3, J_4\})$

(Backward) $(0, v_{sink}, 15, 0, 0, \{\}) \rightarrow (0, v_\star, 15, 0, 0, \{\}) \rightarrow (2, v_{31}, 10, 2, -1, \{J_3\}) \rightarrow (2, v_\star, 10, 2, -1, \{J_3\}) \rightarrow (4, v_{41}, 7, -2, 1, \{J_3, J_4\}) \rightarrow$
$(4, v_\star, 7, -2, 1, \{J_3, J_4\}) \rightarrow (8, v_{11}, 5, 0, -2, \{J_1, J_3, J_4\}) \rightarrow (8, v_\star, 5, 0, -2, \{J_1, J_3, J_4\}) \rightarrow (8, v_{source}, 0, 0, 0, \{J_1, J_3, J_4\})$

To eliminate labels and accelerate the labeling algorithm during the forward and backward search, we perform dominance checks. The following conditions (with at least one inequality or inclusion strictly satisfied) are sufficient to state that a forward label $\vec{L}_1$ dominates a forward label $\vec{L}_2$:

$$
\begin{aligned}
& c^{\vec{L}_1} \geqslant c^{\vec{L}_2} \\
& v^{\vec{L}_1} = v^{\vec{L}_2} \\
& Q_0^{\vec{L}_1} \leqslant Q_0^{\vec{L}_2} \\
& Q_r^{\vec{L}_1} - (Q_0^{\vec{L}_2} - Q_0^{\vec{L}_1}) \leqslant Q_r^{\vec{L}_2} \qquad r \in \mathcal{R} \\
& J^{\vec{L}_1} \subseteq J^{\vec{L}_2}
\end{aligned}
$$

With these conditions, every feasible completion of the partial path associated with $\vec{L}_2$ into a path $\vec{P}_2$ from v_{source} to v_{sink} is also a feasible completion of the partial path associated with $\vec{L}_1$ into a full path $\vec{P}_1$, and the cost of $\vec{P}_2$ is always lower than the cost of $\vec{P}_1$. The third condition considers the possibility of waiting the difference in time between the time associated with $\vec{L}_1$ and the time associated with $\vec{L}_2$ and checks if the secondary resources can be available for a longer time.

Similarly to the forward search, the following conditions are sufficient to state that a backward label $\overleftarrow{L}_1$ dominates a backward label $\overleftarrow{L}_2$:

$$
\begin{aligned}
& c^{\overleftarrow{L}_1} \geqslant c^{\overleftarrow{L}_2} \\
& v^{\overleftarrow{L}_1} = v^{\overleftarrow{L}_2} \\
& Q_0^{\overleftarrow{L}_1} \geqslant Q_0^{\overleftarrow{L}_2} \\
& Q_r^{\overleftarrow{L}_1} + (Q_0^{\overleftarrow{L}_1} - Q_0^{\overleftarrow{L}_2}) \geqslant Q_r^{\overleftarrow{L}_2} \qquad r \in \mathcal{R} \\
& J^{\overleftarrow{L}_1} \subseteq J^{\overleftarrow{L}_2}
\end{aligned}
$$

5.3 Alternative definition of graph G

Dominance checks are performed only between labels ending at the same node. An alternative definition of graph G has therefore a potential impact on the efficiency of solving the RCSPP. We introduce a new graph by removing node $v_\star$ and concatenating every couple of arcs $\big((v_{j,k}, v_\star)(v_\star, v_{j',k'})\big)$ into a single arc. It has the interest of forbidding two consecutive occurrences of the same job directly by the structure of the graph. The introduction of this new graph allows to take into account the dominance rules introduced in Section 3.

We denote $G' = (V', A')$ the new graph with $V' = V \backslash \{v_\star\}$. The set A' is different from A and does not contain any idle arc. For every job $j \in \mathcal{J}$, it contains the arc $(v_{source}, v_{j,k})$ for $k = 1$ and the arc $(v_{j,k}, v_{sink})$ for every k. It also contains an arc $(v_{j,k}, v_{j',k'})$ if job j can be executed in time window T_{jk} and use the main resource immediately before its use by job $j' \neq j$ executed in time window $T_{j'k'}$ (i.e, $t^1_{jk} + p_j \leqslant t^2_{j'k'} - p_{j'}$ if $r_j = r_{j'}$ or $t^1_{jk} + p^-_j + p^0_j \leqslant t^2_{j'k'} - p_{j'} + p^-_{j'}$ if $r_j \neq r_{j'}$). The consumption $q_{a,r}$ of resource r in arc a is defined as follows:

$$
q_{a,r} = \begin{cases}
0 & \text{if } a \in \Gamma^+(v_{source}) \\
p^0_j & \text{if } a \in \Gamma^+(v_{j,k}) \text{ and } r = 0 \\
-p^0_j & \text{if } a \in \Gamma^+(v_{j,k}) \text{ and } r \in \mathcal{R} \backslash \{r_j\} \\
p^-_j + p^+_j & \text{if } a \in \Gamma^+(v_{j,k}) \text{ and } r = r_j
\end{cases}
$$

The accumulated resource consumption interval $[l_{v,r}, u_{v,r}]$ for each node $v \in V'$ and each resource $r \in \mathcal{R}_0$ is identical to the one described for graph G.

The labeling algorithms used to enumerate all feasible paths from v_{source} to v_{sink} in G' are strictly identical to those already presented.

Using the dominance rules introduced in Section 3, we now describe how we reduce the number of arcs of G' while ensuring that at least one of the optimal solutions to the RCSPP is retained. We recall that using an arc $(v_{j,k}, v_{j',k'})$ in a path of graph G' translates into a schedule where the main resource is used by job j during time window T_{jk} and then used by job j' during time window $T_{j'k'}$. The arc $(v_{j',k'}, v_{j,k})$ can be safely removed from G' if the sufficient conditions of Proposition 1 are satisfied. The arc $(v_{j'',k''}, v_{j',k'})$ (resp. $(v_{source}, v_{j',k'})$) can be safely removed if the sufficient conditions stated in item 1 of Proposition 2 are satisfied. Similarly, the arc $(v_{j',k'}, v_{j'',k''})$ (resp. $(v_{j',k'}, v_{sink})$) can be safely removed if the sufficient conditions stated in item 2 of Proposition 2 are satisfied.

We show in Figure 3 the graph G' associated with the instance of Figure 1. The solution of Figure 1 is represented in this graph by the following path: (v_{source}, v_{11}), (v_{11}, v_{41}), (v_{41}, v_{31}), (v_{31}, v_{sink}). Using Proposition 1 with $j = J_1$, $j' = J_3$, $k = 1$ and $k' = 1$, we remove the dotted orange arc (v_{31}, v_{11}). We remove the dashed red arcs using Proposition 2 with $j = J_4$ and j', j'', k' and k'' defined according to the arc to remove. The arcs in gray are removed due to time window incompatibility.

The forward (resp. backward) labels computed when traversing the path forward (resp. backward) are as follows:

(Forward) $(0, v_{source}, 0, 0, 0, \{\}) \rightarrow (0, v_{11}, 2, -2, -2, \{J_1\}) \rightarrow (4, v_{41}, 6, -2, -1, \{J_1, J_4\}) \rightarrow (6, v_{31}, 9, 2, -1, \{J_1, J_3, J_4\}) \rightarrow (8, v_{sink}, 15, 0, 0, \{J_1, J_3, J_4\})$

(Backward) $(0, v_{sink}, 15, 0, 0, \{\}) \rightarrow (2, v_{31}, 10, 2, -1, \{J_3\}) \rightarrow (4, v_{41}, 7, -2, 1, \{J_3, J_4\}) \rightarrow (8, v_{11}, 5, 0, -2, \{J_1, J_3, J_4\}) \rightarrow (8, v_{source}, 0, 0, 0, \{J_1, J_3, J_4\})$

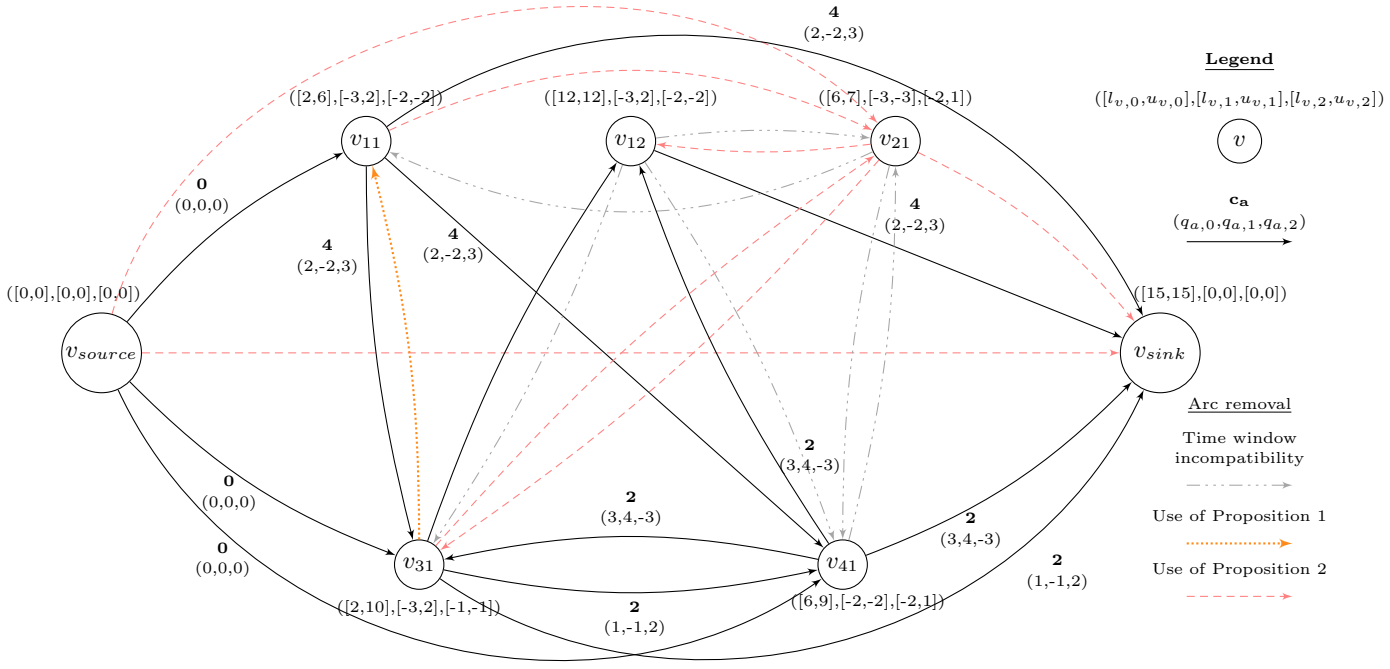


Figure 3: Graph G' associated with the instance of Figure 1

6 A branch-cut-and-price algorithm

Due to the well-known curse of dimensionality, solving the PC-JSOCMSR as a RCSPP using the labeling algorithm described in Section 5 is not practical. The main reason explaining the explosion of the number of labels despite the dominance checks is related to dealing with the elementarity constraints. To overcome this issue, we propose to take these constraints into account in a different way.

We first introduce an extended MILP model for the PC-JSOCMSR. Let $\mathcal{S}$ be the set of schedules satisfying time window restrictions and resource usage constraints. We denote by b_{S} the profit associated with a schedule $\mathsf{S} \in \mathcal{S}$ (sum of the individual profit of the jobs scheduled in S) and by $\tilde{y}_j^{\mathsf{S}} \in \mathbb{N}$ the number of times job j is scheduled in S ($\tilde{y}_j^{\mathsf{S}}$ can be larger than 1 since S does not necessarily satisfy elementarity constraints). Our new model, where the binary variable λ_{S} equal to 1 if and only if schedule S is selected, is as follows:

$$[F3] \quad \max \sum_{\mathsf{S} \in \mathcal{S}} b_{\mathsf{S}} \lambda_{\mathsf{S}} \tag{17}$$

$$\text{s.t.} \quad \sum_{\mathsf{S} \in \mathcal{S}} \tilde{y}_j^{\mathsf{S}} \lambda_{\mathsf{S}} \leqslant 1 \qquad j \in \mathcal{J} \tag{18}$$

$$\sum_{\mathsf{S} \in \mathcal{S}} \lambda_{\mathsf{S}} = 1 \tag{19}$$

$$\lambda_{\mathsf{S}} \in \{0, 1\} \qquad \mathsf{S} \in \mathcal{S} \tag{20}$$

The objective function (17) maximizes the total profit associated with the selected schedules. Constraints (18) impose the elementarity constraints. Constraint (19) imposes the selection of a single schedule. Constraints (20) set the domains of the decision variables.

We introduce a branch-cut-and-price (BCP) algorithm to solve formulation $[F3]$. Since the number of variables is exponential in the number of jobs, we apply column generation to solve its linear relaxation. The variables λ are dynamically generated by solving the pricing problem described in §6.2 and defined as a RCSPP on graph G or G' (with some adjustments). To tighten the dual bound, we generate cuts derived from Chvátal-Gomory rank-1 cuts (21). This is explained in §6.1. The whole column and cut generation procedure is detailed in §6.3. We finally describe in §6.4 the branching strategy we use to discard non-integer solutions.

6.1 Chvátal-Gomory rank-1 cuts

Let $J \subseteq \mathcal{J}$ and $(\mu_j)_{j \in J}$ be a vector of strictly positive multipliers, a Chvátal-Gomory rounding of set packing constraints (18) results in the following Chvátal-Gomory rank-1 cut (R1C)

$$\sum_{\mathsf{S} \in \mathcal{S}} \lfloor \sum_{j \in J} \mu_j \tilde{y}_j^{\mathsf{S}} \rfloor \lambda_{\mathsf{S}} \leqslant \lfloor \sum_{j \in J} \mu_j \rfloor \tag{21}$$

that is a valid inequality for $[F3]$. R1Cs are non-robust in the sense they impact the structure of the pricing problem (due to the floor rounding operation). However, their separation can improve the quality of the Lagrangian dual bound. As a solution to $[F3]$ is a single schedule, we only consider R1Cs such that $\lfloor \sum_{j \in J} \mu_j \rfloor < 1$ (i.e., $|J| = 1$) as other R1Cs are practically never violated due to constraint (19). Specifically, we separate the following R1Cs:

$$\sum_{\mathsf{S} \in \mathcal{S}} \lfloor 0.5 \tilde{y}_j^{\mathsf{S}} \rfloor \lambda_{\mathsf{S}} \leqslant 0 \qquad j \in \mathcal{J} \tag{22}$$

These cuts can be violated by the linear relaxation solution since the schedules in $\mathcal{S}$ do not necessarily satisfy the elementarity constraints.

To mitigate the impact of these cuts on the pricing problem difficulty, we consider the limited memory version of R1Cs (22) introduced by Pecin et al. (2017b). Each R1C is associated with an arc memory set $AM \subseteq A$ as described in Pecin et al. (2017a). With the arc memory set, the coefficient $\lfloor 0.5 \tilde{y}_j^{\mathsf{S}} \rfloor$ of variable λ_{S} is replaced by a smaller coefficient $\varphi(j, AM, \mathsf{S})$. The limited-arc-memory R1C cut associated with job $j \in \mathcal{J}$ reads

as follows:

$$\sum_{\mathsf{S}\in\mathcal{S}} \varphi(j, AM, \mathsf{S})\lambda_{\mathsf{S}} \leqslant 0 \tag{23}$$

Algorithm 1 describes the computation of $\varphi(j, AM, \mathsf{S})$. The variable e is referred to as the *state* variable. Depending on its value, earlier occurrences of job j may be forgotten when an arc does not belong to the memory.

Algorithm 1: Computing $\varphi(j, AM, \mathsf{S})$

```
φ ← 0, e ← 0
foreach arc a of schedule S do
    if a ∉ AM then e ← 0
    if a ∈ ⋃_{v∈V_j} Γ⁻(v) then
        e ← e + 0.5
        if e = 1 then φ ← φ + 1, e ← 0
    end
end
return φ
```

The arc memory set AM is computed as the minimal subset of arcs that ensure that the coefficient in (23) of each variable λ_{S} with a positive value in the current linear relaxation solution to $[F3]$ is equal to its coefficient in (22) (see Pecin et al. (2017b) for details). If an arc (v, v') is added to AM, then the arc (v', v) is also added to AM.

Using a limited memory version of R1Cs implies that a cut of type (22) for a job j may be violated by the current linear relaxation solution although a limited memory version of this cut exists. This means that a different arc memory set AM' must be used to cut off the solution. In that case, we create a new cut combining AM' and the arc memory sets of existing cuts associated with j.

6.2 Pricing problem

Let $(\pi_j)_{j\in\mathcal{J}}$ be the dual variables associated with constraints (18), ζ be the dual variable associated with constraints (19), and $(\sigma_g)_{g\in\mathcal{G}}$ be the dual variables associated with active R1Cs. We denote by $j(g)$ the job associated with R1C $g \in \mathcal{G}$ and $AM(g)$ its arc memory set. The reduced cost of a schedule S is equal to:

$$\bar{c}_{\mathsf{S}} = \sum_{j\in\mathsf{S}}(b_j - \pi_j) - \sum_{g\in\mathcal{G}} \varphi(j(g), AM(g), \mathsf{S})\sigma_g - \zeta \tag{24}$$

The pricing problem can be stated as: $\max_{\mathsf{S}\in\mathcal{S}}\{\bar{c}_S\}$.

We define the pricing subproblem as a RCSPP on the directed graph $G = (V, A)$ (or indifferently on $G' = (V', A')$) introduced in Section 5. The cost c_a of an arc a is replaced by $\bar{c}_a$ its reduced cost defined as:

$$\bar{c}_a = \begin{cases} b_j - \pi_j & \text{if } a \in \Gamma^+(v_{j,k}) \\ 0 & \text{otherwise} \end{cases}$$

Introducing $\tilde{x}_a^{\mathsf{S}}$ as a parameter equal to the number of times arc a belongs to the path in one-to-one correspondence with schedule S, we can rewrite (24) as follows:

$$\bar{c}_{\mathsf{S}} = \sum_{a\in A} \tilde{x}_a^{\mathsf{S}}\bar{c}_a - \sum_{g\in\mathcal{G}} \varphi(j(g), AM(g), \mathsf{S})\sigma_g - \zeta \tag{25}$$

We solve the pricing problem applying the bi-directional algorithm described in §5.1 with the adaptions of §5.2 for generating paths only associated with semi-active schedules. We now describe additional modifications that are required, in particular in the definition of the labels, their extension, and the dominance checks. Pseudo-code of these algorithms are available in §S.2 of the supplementary material.

Forward labeling algorithm The forward labeling algorithm uses a forward label $\vec{L} = \left(\bar{c}^{\vec{L}}, v^{\vec{L}}, Q^{\vec{L}}, E^{\vec{L}}\right)$ to represent a path starting at v_{source} and finishing at node $v^{\vec{L}} \in V$ with accumulated resource consumption $Q^{\vec{L}}$ and a reduced cost of $\bar{c}^{\vec{L}}$ (in the place of $c^{\vec{L}}$). Forward label $\vec{L}$ is also characterized by a vector of states $E^{\vec{L}}$ (see the state variable introduced in §6.1) associated with R1Cs active in the current linear relaxation solution (the set $J^{\vec{L}}$ is dropped). When extending the label $\vec{L}$ along an arc $a = (v^{\vec{L}}, v)$, the resulting label $\vec{L}'$ has the following characteristics (only differences are described):

$$\bar{c}^{\vec{L}'} = \bar{c}^{\vec{L}} + \bar{c}_a - \sum_{g \in \mathcal{G}} [a \in AM(g) \wedge v \in V_{j(g)} \wedge E_g^{\vec{L}} + 0.5 = 1]\sigma_g$$

$$E_g^{\vec{L}'} = \begin{cases} 0 & \left(a \notin AM(g) \wedge v \notin V_{j(g)}\right) \vee \left(AM(g) \wedge v \in V_{j(g)} \wedge E_g^{\vec{L}} + 0.5 = 1\right) \\ 0.5 & a \notin AM(g) \wedge v \in V_{j(g)} \\ E_g^{\vec{L}} & a \in AM(g) \wedge v \notin V_{j(g)} \\ E_g^{\vec{L}} + 0.5 & a \in AM(g) \wedge v \in V_{j(g)} \wedge E_g^{\vec{L}} + 0.5 < 1 \end{cases} \qquad g \in \mathcal{G}$$

where the square brackets are Iverson brackets ($[\mathsf{P}] = 1$ if statement P is true, and $[\mathsf{P}] = 0$ otherwise). When checking if a forward label $\vec{L}_1$ dominates another forward label $\vec{L}_2$, we replace the first condition (related to the label costs) by:

$$\bar{c}^{\vec{L}_1} \geqslant \bar{c}^{\vec{L}_2} + \sum_{g \in \mathcal{G}: E_g^{\vec{L}_1} > E_g^{\vec{L}_2}} \sigma_g$$

Backward labeling algorithm The backward labeling algorithm uses a backward label $\overleftarrow{L} = (\bar{c}^{\overleftarrow{L}}, v^{\overleftarrow{L}}, Q^{\overleftarrow{L}}, E^{\overleftarrow{L}})$ to represent a (backward) path starting at v_{sink} and finishing at node $v^{\overleftarrow{L}} \in V$. When extending the label $\overleftarrow{L}$ along an arc $a = (v, v^{\overleftarrow{L}})$, the resulting label $\overleftarrow{L}'$ has the following new characteristics:

$$\bar{c}^{\overleftarrow{L}'} = \bar{c}^{\overleftarrow{L}} + \bar{c}_a - \sum_{g \in \mathcal{G}} [a \in AM(g) \wedge v \in V_{j(g)} \wedge E_g^{\overleftarrow{L}} + 0.5 = 1]\sigma_g$$

$$E_g^{\overleftarrow{L}'} = \begin{cases} 0 & \left(a \notin AM(g) \wedge v \notin V_{j(g)}\right) \vee \left(AM(g) \wedge v \in V_{j(g)} \wedge E_g^{\overleftarrow{L}} + 0.5 = 1\right) \\ 0.5 & a \notin AM(g) \wedge v \in V_{j(g)} \\ E_g^{\overleftarrow{L}} & a \in AM(g) \wedge v \notin V_{j(g)} \\ E_g^{\overleftarrow{L}} + 0.5 & a \in AM(g) \wedge v \in V_{j(g)} \wedge E_g^{\overleftarrow{L}} + 0.5 < 1 \end{cases} \qquad g \in \mathcal{G}$$

When checking if a backward label $\overleftarrow{L}_1$ dominates another backward label $\overleftarrow{L}_2$, we replace the first condition (related to the label costs) by:

$$\bar{c}^{\overleftarrow{L}_1} \geqslant \bar{c}^{\overleftarrow{L}_2} + \sum_{g \in \mathcal{G}: E_g^{\overleftarrow{L}_1} > E_g^{\overleftarrow{L}_2}} \sigma_g$$

Using the same reasoning as in §5.1, the time and space complexity of the labeling algorithms is pseudo-polynomial in the worst case if $\mathcal{G} = \varnothing$, while otherwise it is exponential.

Bi-directional labeling algorithm The last condition (related to elementarity constaints) for concatenating a forward path with a backward path is dropped. The reduced cost of a path obtained as the concatenation of a (partial) path represented by a forward label $\vec{L}$ and a (partial) backward path represented by a backward label $\overleftarrow{L}$ along an arc $a = (v^{\vec{L}}, v^{\overleftarrow{L}})$ is equal to $\bar{c}^{\vec{L}} + \bar{c}_a + \bar{c}^{\overleftarrow{L}} - \sum_{g \in \mathcal{G}: E_g^{\vec{L}} + E_g^{\overleftarrow{L}} \geqslant 1} \sigma_g$.

Bucket graph We use a bucket graph implementation of the labeling algorithms as proposed in (Sadykov et al., 2021). Labels are grouped and stored in buckets to improve the efficiency of the dominance checks. At each node v, we create $u_{v,0} - l_{v,0} + 1$ buckets, each bucket being associated with a specific time. Every label corresponding to a partial path ending up at the same node and with the same accumulated resource consumption of the main resource (i.e., with the same time) are stored in the same bucket. Using buckets helps reducing the number of dominance checks and the computational time needed for solving the pricing problem.

Dominance checks are performed only with labels within the same bucket when testing the insertion of a new label. Before extending a label, dominance checks are performed by considering the labels stored in all buckets created at a node and associated with a lower time. Let us denote each bucket as a couple (v,t) of a node $v \in V$ and a time $t \in [\![l_{v,0}, u_{v,0}]\!]$. In the forward (resp. backward) bucket graph implementation of G, for every bucket (v,t) and every arc $a = (v,v')$ (resp. $a = (v',v)$), we create an arc between bucket (v,t) and bucket $(v', \max\{l_{v',0}, t+q_{a,0}\})$ (resp. an arc between bucket (v,t) and bucket $(v', \min\{u_{v',0}, t-q_{a,0}\})$. The bucket graph implementation of the graphs G and G' displayed in Figures 2 and 3 are given in §S.3 of the supplementary material. The bucket graph is used for the reduced cost-based arc elimination procedure described in §6.3.

6.3 Column and cut generation algorithm

We use an adaptation of the BCP algorithm introduced by Sadykov et al. (2021). To make the paper self-contained, we describe the main components of this algorithm.

For each node of the branch-and-bound tree, we compute the linear relaxation of the corresponding restricted master program using a three-stage column generation procedure. The first two stages use heuristic pricing and generates at most 30 columns per call, whereas the last stage uses exact pricing and generates at most 150 columns per call. During the first stage, the pricing problem is solved by keeping only in every bucket the eight labels with the largest reduced cost. During the second phase, the dominance checks do not include the conditions on the R1Cs. The last stage performs full dominance checks. After each pricing in this stage, the threshold value t^* for the bidirectional labeling algorithm (initially set equal to $(t^1 + t^2)/2$), is automatically adjusted as follows: if the number of non-dominated labels in the forward (resp. backward) algorithm is larger than the number of non-dominated labels in the backward (resp. forward) algorithm, then t^* is increased (resp. decreased) by 5%.

To improve the column generation convergence, we use the automatic dual price smoothing technique described in Pessoa et al. (2018) in each stage.

When the number of columns in the restricted master program exceeds 10000, we remove 34% of the columns by selecting the non-basic columns whose reduced cost is the lowest.

When column generation converges, we eliminate arcs of graph G (or G') based on reduced costs. If the maximum reduced cost of a path using an arc $a \in A$ is smaller than the difference between the primal bound and the dual bound, then a can be removed from G as it does not belong to any improving solution to $[F3]$ (see Irnich et al. (2010)). The value of a feasible heuristic solution to the PC-JSOCMSR is given as input and initializes the value of the best primal bound. To test if an arc $a = (v,v')$ should be removed, we test all the combinations of a forward label $\vec{L}$ such that $v^{\vec{L}} = v$ with arc a and a backward label $\overleftarrow{L}$ such that $v^{\overleftarrow{L}} = v'$ that lead to a feasible path. As the labeling algorithms described in §6.2 use the bucket graph implementation of Sadykov et al. (2021), we apply the *bucket arc elimination procedure* described in this latter work. Specifically, using the condition previously described, we remove arcs in the bucket graph implementation of G (or G') rather than in G (or G'). For the validity of the algorithm, we need to introduce "*jump*" arcs between the buckets of the same node associated with successive times. A jump arc a between bucket (v,t) and bucket (v,t') is such that $c_a = 0$, $q_{a,0} = t' - t$, and $q_{a,r} = t - t'$ for every $r \in \mathcal{R}$. Labels are also extended along these arcs. The bucket arc elimination procedure is stronger than a classical arc elimination procedure in G, since an arc of G can only be eliminated based on reduced cost only if all the arcs associated with it in the bucket graph implementation of G can be eliminated. The arc elimination procedure is performed after the initial convergence and also each time the current dual-primal gap decreased by at least 10% since it was last applied.

We separate the R1Cs (22) by enumeration. In each cut generation round, we consider at most 50 R1Cs (the most violated in priority) and add the limited memory version (23) of these cuts.

Column and cut generation is stopped either when after 3 rounds of cut generation the dual-primal gap decreases by less than 2% per round (tailing-off condition), or when the time spent to solve the pricing subproblem exceeds 5 seconds (time-efficiency condition).

If the current dual-primal gap is less than 1%, we try to solve the pricing problem using the bidirectional labeling algorithm described in §5.2 and the additional constraint that the reduced cost should be larger than

the difference between the primal bound and the dual bound (reduced costs are considered like in §6.2). For efficiency consideration, we stop the algorithm if more than $1,000,000$ forward (backward) labels are generated. If the algorithm succeeds (i.e., it has not been stopped), the pricing subproblem is solved by inspection in future column generation iterations. If the total number of enumerated paths is less than $10,000$, all the paths are added to $\mathcal{S}$ and formulation $[F3]$ is solved with a MILP solver. An optimal solution at the current node is obtained, and this node can be pruned (after updating the current best primal bound if necessary). Otherwise, we proceed to branching. The branch-and-bound search tree is explored using a depth-first search strategy. Among the nodes with same depth, the algorithm selects the one with the largest dual bound.

6.4 Branching

To reach integrality, we perform branching on the value of the following aggregated variables:

- the number of times a job j is performed:

$$\sum_{\mathsf{S}\in\mathcal{S}} \tilde{y}_j^{\mathsf{S}}\lambda_{\mathsf{S}} = \sum_{\mathsf{S}\in\mathcal{S}} \sum_{a\in \bigcup_{v\in V_j}\Gamma^+(v)} \tilde{x}_a^{\mathsf{S}}\lambda_{\mathsf{S}}$$

- the number of times a job j is performed during a particular time window $k \in [\![1, w_j]\!]$:

$$\sum_{\mathsf{S}\in\mathcal{S}} \sum_{a\in\Gamma^+(v_{j,k})} \tilde{x}_a^{\mathsf{S}}\lambda_{\mathsf{S}}$$

- the number of times arc $a \in A$ is used:

$$\sum_{\mathsf{S}\in\mathcal{S}} \tilde{x}_a^{\mathsf{S}}\lambda_{\mathsf{S}}$$

Adding a branching constraint to the (restricted) master linear program does not make the pricing problem harder to solve. The dual variable ψ associated with a generic branching constraint written as $\sum_{\mathsf{S}\in\mathcal{S}}\sum_{a\in A} h_a\tilde{x}_a^{\mathsf{S}}\lambda_{\mathsf{S}} \leqslant H$ is considered in the pricing problem by subtracting $h_a\psi$ from $\bar{c}_a$ for every $a \in A$.

After performing the previous branching strategy, we may end up with a fractional solution. In that case, we branch over accumulated consumption of resource 0 (Pessoa et al., 2021). Let Q_a^{S} be the accumulated consumption of resource 0 at the origin of arc a in schedule S (i.e., the time at which the current occurrence of the job associated with a starts using the main resource). We compute for each job j the value t_j^* such that $\sum_{\mathsf{S}\in\mathcal{S}}\sum_{a\in\bigcup_{v\in V_j}\Gamma^+(v)}[Q_a^{\mathsf{S}} \leqslant t_j^*]\lambda_{\mathsf{S}}$ is the most fractional. If there exists a threshold t_j^* for a job j leading to a fractional value, then we create two branches: in the left branch we set $u_{v,0} = t_j^*$ for every $v \in V_j$ such that $u_{v,0}$ is larger than t_j^*, and in the right branch we set $l_{v,0} = t_j^*$ for every $v \in V_j$ such that $l_{v,0}$ is less than t_j^*.

We use the three-phase evaluation procedure described by Sadykov et al. (2021) to choose the most promising branching candidate among the aggregated variable (presented above) with a fractional value. This branching procedure is inspired by strong branching. First, it chooses at most 30 branching candidates: half are chosen based on the history (if not empty) of previous calls to the branching procedure, and the remaining candidates are chosen according to the distance between their fractional value and the nearest integer. Second, it solves the LP relaxation of the restricted master problem modified with the corresponding branching constraint. Third, the procedure selects the three candidates with the largest product of dual bound improvements in the two branches, and performs column generation using heuristic pricing. The candidate with the largest product of dual bound improvements in the two branches is selected for branching.

7 A heuristic algorithm

We introduce a new heuristic algorithm for the PC-JSOCMSR with two goals. First, we aim at providing a good initial solution quickly to the BCP algorithm as it is essential to speed-up its convergence (for arc elimination

and nodes pruning). Second, we aim at computing high quality solutions for large-sized instances that remain beyond the reach of an exact algorithm.

Our heuristic is an iterated local search (ILS) algorithm. ILS is a metaheuristic that iteratively applies a local search phase to produce local optima, and a perturbation mechanism to escape from them. We use a large neighborhood search (LNS) for the local search phase. Algorithm 2 outlines the general structure of our heuristic. First, we build a solution s using the LNS repair (insertion) operator (see §7.1). The algorithm then enters an iterative process. At every iteration, the LNS phase (see §7.1) starts from solution s and accepts only improving solutions. It stops when either the number of iterations reaches δ^{max} or the number of iterations without improvements reaches θ^{max}. If appropriate, the algorithm updates the current best solution $s^\star$. Then, the algorithm performs the perturbation phase (see §7.2) to avoid being trapped in a local optimum that cannot be escaped (or escaped in a reasonable number of iterations) by successive application of the destroy and reparator operators. It modifies $s^\star$ to produce a new solution s for the next LNS phase with the aim of exploring a new region of the search space. We only accept the solution produced by the perturbation operator if its relative gap with respect to the best solution computed so far is less than or equal to ϵ. This is to avoid too much re-optimization and to increase the possibility to find improving solutions with the LNS. The algorithm stops when a total of δ^{max} iterations of the LNS have been performed out of all iterations. The remainder of this section describes the LNS and perturbation phases.

Algorithm 2: ILS

input: two positive integer values δ^{max} and θ^{max}, a float value ϵ
Build s^0 using the LNS repair operator ▷ Initialization (see §7.1)
$\delta \leftarrow 0$, $s^\star \leftarrow s^0$, $s \leftarrow s^0$
while $\delta < \delta^{max}$ **do**
  $\theta \leftarrow 0$
  while $\delta < \delta^{max} \wedge \theta < \theta^{max}$ **do** ▷ LNS phase (see §7.1)
    $s' \leftarrow \texttt{repair}(\texttt{destroy}(s))$
    if $f(s') > f(s)$ **then** ▷ *$f(s)$ denotes the value of the objective function for a solution s*
      $s \leftarrow s'$
    else
      $\theta \leftarrow \theta + 1$
    end
    $\delta \leftarrow \delta + 1$
  end
  if $f(s) > f(s^\star)$ **then** $s^\star \leftarrow s$;
  repeat ▷ Perturbation phase (see §7.2)
    $s \leftarrow \texttt{perturb}(s^\star)$
  until $1 - f(s)/f(s^\star) \leqslant \epsilon$;
end
return $s^\star$

7.1 The LNS phase

LNS is a local search algorithm where the current solution is successively partially destroyed and then repaired. Our LNS uses a single destroy and repair operator. It explores the solution space without allowing any violation of the time constraints (nor other constraints). It has two stopping conditions: a maximum total number of iterations and a maximum number of iterations without improvement. Its implementation borrow ideas from the iterated local search introduced by Vansteenwegen et al. (2009) to solve the TOPTW.

Our LNS relies on the modeling of a solution to the PC-JSOCMSR as a sequence of jobs using the main resource. To reduce the complexity and increase the time efficiency of testing the feasibility of operations modifying the solution, we maintain predecessor and successor information on each resource for each scheduled job. Let $\mathcal{J}^{sch}$ the set of jobs scheduled during the planning horizon in the current solution. We associate with each job $j \in \mathcal{J}^{sch}$ its predecessor $pred_j^r$ and successor $succ_j^r$ on resource $r \in \mathcal{R}$. We introduce two dummy jobs denoted by *head* and *tail* and define them as the predecessor of the first scheduled job and successor of the last scheduled job on every resource. The processing time attributes associated with these dummy jobs

are set to 0. We associate with each scheduled job its starting time on the main resource (denoted by $start_j^0$) and an attribute (denoted by $shift_j$) representing the maximum value by which $start_j^0$ can be shifted while still maintaining time window feasibility for the subsequent jobs. We also set $start_{head}^0 = t^1$, $start_{tail}^0 = t^2$, $shift_{head} = shift_{tail} = 0$. For better readability and to ease the presentation of the algorithm, we hereafter denote by $start_j$, end_j, and end_j^0 the starting time of job j, its ending time on the secondary resource and its ending time on the main resource, respectively. Specifically, we have $start_j = start_j^0 - p_j^-$, $end_j = start_j + p_j$, and $end_j^0 = start_j^0 + p_j^0$. Note that we only work with semi-active schedules in the algorithm, meaning that a job is scheduled after the end of the previous job on resource 0 as early as possible with respect to the time windows. The attributes are computed as follows and updated throughout the algorithm:

$$start_j^0 = \begin{cases} t_{j1}^1 + p_j^- & \text{if } pred_j^0 = head \\ \phi_j^{es}\big(end_{pred_j^0}^0, end_{pred_j^{r_j}}\big) + p_j^- & \text{if } pred_j^0 \neq head \end{cases}$$

$$shift_j = \begin{cases} t_{jw_j}^2 - (start_j^0 + p_j^0 + p_j^+) & \text{if } succ_j^0 = tail \\ \phi_j^{ls}\big(start_{succ_j^0}^0 + shift_{succ_j^0}, start_{succ_j^{r_j}} + shift_{succ_j^{r_j}}\big) & \text{if } succ_j^0 \neq tail \end{cases}$$

where $\phi_j^{es}(t,t')$ denotes the earliest starting time possible for j if the resources 0 and r_j become available at time t and t', and $\phi_j^{ls}(t,t')$ denotes the latest starting time possible for j if the resources 0 and r_j should become available at time t and t' (see their mathematical definition in §S.1.2 of the supplementary material).

The heuristic relies on a destroy operator that removes jobs from the current solution (i.e., from $\mathcal{J}^{sch}$). This destroy operator works with a position $l \in [\![1, |\mathcal{J}^{sch}|]\!] = \{c \in \mathbb{N} : 1 \leqslant c \leqslant |\mathcal{J}^{sch}|\}$ in the sequence of jobs scheduled in the current solution and a value $\beta \in [\![\beta^{min}, \beta^{max}]\!]$ representing the number of jobs to remove from that position. If there are less than β jobs scheduled after position l, then it removes jobs from the beginning of the time horizon. The rationale behind it is that contiguous jobs and contiguous free space increase the possibility for insertion of unscheduled jobs. The value of l is incremented by one every time an improving solution has not been found at the last iteration. If l reaches the end of the sequence (i.e., $l = |\mathcal{J}^{sch}|$), then l is reset to 1 and β is incremented by one if it is below its maximum possible value β^{max}.

At the end of the destroy phase, we also apply with a probability of 0.5 a random perturbation to the solution. Specifically, we replace η tasks by non-scheduled tasks from $\mathcal{J} \backslash \mathcal{J}^{sch}$. The selection among feasible replacements is performed using a uniform probability distribution.

After jobs removal and a potential perturbation, we use a repair operator to build a new solution. This operator consider two operations: *replacement* and *insertion*. A replacement consists of replacing a scheduled job by an unscheduled job, while an insertion consists of inserting an unscheduled job into the solution. Replacing job j_1 by job j_2 is feasible if the following conditions are satisfied:

$$\begin{cases} \phi_j^{es}\big(end_{pred_{j_1}^0}^0, end_{pred_{j_1}^{r_{j_2}}}^{r_{j_2}}\big) + p_{j_2}^- + p_{j_2}^0 \leqslant start_{succ_{j_1}^0}^0 + shift_{succ_{j_1}^0} \\ \phi_j^{es}\big(end_{pred_{j_1}^0}^0, end_{pred_{j_1}^{r_{j_2}}}^{r_{j_2}}\big) + p_{j_2} \leqslant start_{succ_{j_1}^{r_{j_2}}}^{r_{j_2}} + shift_{succ_{j_1}^{r_{j_2}}} \end{cases}$$

We check if the shift induced by the operation to the subsequent jobs on the main and secondary resources is feasible without any time violation. If this is the case, we set $start_{j_2}^0 = \phi_j^{es}\big(end_{pred_{j_1}^0}^0, end_{pred_{j_1}^{r_{j_2}}}^{r_{j_2}}\big)$ and update $\mathcal{J}^{sch}$ to $\big(\mathcal{J}^{sch} \backslash \{j_1\}\big) \cup \{j_2\}$. Inserting job j_2 after job j_1 is feasible if the following conditions (similar to the conditions for a replacement) are satisfied:

$$\begin{cases} \phi_j^{es}\big(end_{j_1}^0, end_{pred_{succ_{j_1}^0}^{r_{j_2}}}^{r_{j_2}}\big) + p_{j_2}^- + p_{j_2}^0 \leqslant start_{succ_{j_1}^0}^0 + shift_{succ_{j_1}^0} \\ \phi_j^{es}\big(end_{j_1}^0, end_{pred_{succ_{j_1}^0}^{r_{j_2}}}^{r_{j_2}}\big) + p_{j_2} \leqslant start_{succ_{j_1}^{r_{j_2}}}^{r_{j_2}} + shift_{succ_{j_1}^{r_{j_2}}} \end{cases}$$

If this is the case, we set $start_{j_2}^0 = \phi_j^{es}\big(end_{j_1}^0, end_{pred_{succ_{j_1}^0}^{r_{j_2}}}^{r_{j_2}}\big)$ and update $\mathcal{J}^{sch}$ to $\mathcal{J}^{sch} \cup \{j_2\}$.

We compute a score for each feasible operation. The score w_o of an operation o corresponding to the insertion of job j after a currently scheduled job j' is equal to

$$w_o = \frac{m}{m+1} \times b_j \times \Big(\frac{1}{p_j^0 + \alpha.wait^0} + \frac{1}{m(p_j + \alpha.wait)}\Big)$$

with $wait_0 = \min\big\{\max\{0, start^0_{succ^0_j} - end^0_j\}, \max\{0, start^0_j - end^0_{pred^0_j}\}\big\}$ and $wait = \min\big\{\max\{0, start_{succ^{r_j}_j} - end_j\}, \max\{0, start_j - end_{pred^{r_j}_j}\}\big\}$. The score w_o of an operation o corresponding to the replacement of job j' at its current position by job j is equal to

$$w_o = \frac{m}{m+1} \times (b_j - b_{j'}) \times \Big(\frac{1}{p^0_j + \alpha.wait^0} + \frac{1}{m(p_j + \alpha.wait)}\Big)$$

The score of an operation increases as the inserted job has a large profit by time unit of use of the main resource and the secondary resource. The use of the secondary resource is considered with a factor equal to $1/m$. Inserting a job that leads to an important waiting time with respect to both its predecessor and successor is penalized with a factor equal to $\alpha \in [0, 1]$. The algorithm selects the operation to apply among the γ operations with the highest score using a uniform probability distribution. It continues as long as there exist an operation with a strictly positive score. This means that a replacement can only be applied if it improves the objective function of the solution. This avoids a cycling effect, but may restrict the search space. The random perturbation applied before the reparation phase is meant to alleviate this limitation, since non-improving replacements can be applied. Note that the removed jobs are candidates for an insertion at a different position in the solution. These jobs may end up at their initial position at the end of the repair procedure, but only if it is profitable for the objective function.

After any modification of the current solution, we recompute the solution attributes requiring an update. Specifically, if j' is the first job scheduled after a sequence of consecutive jobs that has been removed or after the job that has been inserted (either by an insertion or a replacement operation), we update $start^0_j$ to $\overline{start}^0_j$ and $shift_j$ to $\overline{shift}_j$ as follows:

$$\overline{start}^0_j = \begin{cases} start^0_j & \text{if } j \in \{head, tail\} \vee j \text{ scheduled before } j' \text{ on the main resource} \\ \phi^{es}_j\big(\overline{end}^0_{pred^0_j}, \overline{end}_{pred^r_j}\big) & \text{otherwise} \end{cases}$$

$$\overline{shift}_j = \begin{cases} shift_j - (\overline{start}^0_j - start^0_j) & \text{if } j = j' \vee \big(j \neq tail \wedge j \text{ scheduled after } j' \text{ on the main resource}\big) \\ \phi^{ls}_j\big(\overline{start}^0_{succ^0_j} + \overline{shift}_{succ^0_j}, \overline{start}_{succ^r_j} + \overline{shift}_{succ^r_j}\big) - \overline{start}^0_j & \text{otherwise} \end{cases}$$

We start by computing the attributes forward from j' to *tail* and then backward from $pred^0_{j'}$ to *head*. We stop the update in each direction as soon as the starting time of the last job considered by the update procedure has not changed on every resource.

7.2 The perturbation phase

To escape from a local optimum, we perturb the current best solution $s^\star$ by removing β^{max} jobs using a uniform probability distribution. Contrary to the repair operator, the jobs removed are not necessarily consecutive. Next, we replace randomly η scheduled jobs by non-scheduled ones. We finally use the LNS repair operator described in the previous section to improve the solution. We want the resulting solution to have similarities to $s^\star$, but enough differences to explore a different part of the search space in the next LNS phase.

8 Computational experiments

We coded all the algorithms using C++ and compiled them with GCC 9.2.0. We used the BaPCod C++ library v.053 (Sadykov and Vanderbeck, 2021) for the implementation of the BCP algorithm. For the bucket graph implementation of the labeling algorithm, path enumeration, and the separation of limited-memory R1Cs, we used the C++ code developed by Sadykov et al. (2021). Modifications to these generic codes have been required to handle specific characteristics of the PC-JSOCMSR, especially for the labeling algorithms. We used IBM CPLEX Optimizer version 12.10.0 with the default parameters as the LP solver in column generation, and as the MILP solver for the formulations presented in Section 4. The experiments were run on a 2 Deca-core Ivy-Bridge Haswell Intel Xeon E5-2680 v3 server running at 2.50 GHz. The 128 GB of available RAM was

shared between six copies of the algorithm running in parallel on the server. Each instance is solved by one copy of the algorithm using a single thread.

We considered the benchmark instances[1] used in Horn et al. (2021a). The instances are split into two sets. The total number of jobs n in these instances belongs to $\{50, 100, 150, 200, 250, 300, 400, 500\}$ (we excluded the instances with 350 and 450 to reduce the number of tests). The set denoted A contains instances related to the pre-runtime scheduling of avionic systems. It has the following characteristics: the length of the planning horizon is 1000, the number of secondary resources m is equal to three or four, the number of jobs that can be scheduled within their time windows is independent of the total number of jobs, "regular" jobs require the main resource either at the beginning or at the end of their processing and for between one fifth and one eighth of their total processing time (that is between 200 and 320 for "regular" jobs), "communication" jobs have a processing time on the main and secondary resource equal to 40, between 20 to 30 jobs can be scheduled during the planning horizon, and the prize of each job refers to a priority. The set denoted P are instances related to particle therapy scheduling. Their characteristics are the following: the length of the planning horizon (from 100 to 800) depends on the number of jobs, the number of secondary resources m is equal to two or three, the number of jobs that can be scheduled within their time windows grows approximately linearly with the total number of jobs, the jobs require the main resource on average during the second third of their processing, the average processing time of a job is 40, around 30% of the jobs can be scheduled during the planning horizon, and the prize of each job is correlated to its processing time on the main resource. In all the instances of set A and P, the secondary resources are equally distributed among the jobs. Jobs have between one and three time windows (except the communication jobs in set A which can have up to six time windows). There are 30 instances in each set for each combination of n and m, which makes a total of 960 instances. We refer to the Appendix D in Horn et al. (2021a) for details about the generation procedure.

The first aim of our computational experiments is to assess the performance of a commercial MILP solver for solving instances of the PC-JSOCMSR using the compact and pseudo-polynomial formulations introduced in section 4. These results are presented in §8.1. The second aim of our experiments is to assess the effectiveness of our heuristic algorithm introduced in section 7 to build high-quality solutions to the PC-JSOCMSR. The results and comparison with the literature are presented in §8.2. The third and most important aim of our computational experiments is to assess the quality of our BCP algorithm as an exact solution method for the PC-JSOCMSR. We also compare with results from the literature. We report in §S.4 of the supplementary material the results for each combination of instance parameters, meaning that each line is the average of the results of 30 instances (60 instances when reporting the results either according to the number of jobs or the number of secondary resources). The detailed results for every single tested instance are available online[2]. In all the tables, the CPU time is given in seconds.

8.1 Results for [F1] and [F2] with a MILP solver

We empirically tested the performance of solving formulations $[F1]$ and $[F2]$ with a MILP solver. We restrict our analysis to the instances with 50 and 100 jobs. We ran the two MILP formulations with a two-hour CPU time limit (single thread). For each formulation, Table 1 reports the number of instances with a solution proven to be optimal (#Opt), the average CPU time in seconds (Time), the average gap (Gap) with respect to the upper bound retrieved by the solver. For a better comparison, we also report the average gap ($\text{Gap}^{[F2]}$) with respect to the upper bound retrieved by the solver on the model $[F2]$ (this bound is always lower than the bound obtained with the model $[F1]$). We compute the gap with respect to a value $\bar{z}$ as $(\bar{z} - z)/\bar{z}$, where z is the objective function value of the best integer solution returned by the solver. For unsolved instances, the computational time is set to the time limit.

We observe that the MILP solver provides proven optimal solutions for all the instances when using model $[F2]$, whereas it usually fails to return optimal solutions within the time limit when using model $[F1]$. In the

[1] Available from `https://www.ac.tuwien.ac.at/research/problem-instances/` (last accessed: 2021/07/13)

[2] `https://www.math.u-bordeaux.fr/~afroger001/documents/Results_PCJSOCMSR_Article.tar.xz`

			[F1]				[F2]		
Set	n	m	#Opt	Time	Gap	Gap$^{[F2]}$	#Opt	Time	Gap
A	50	3	0/30	7200	15.2%	0.5%	30/30	6	0.0%
		4	0/30	7200	19.2%	0.8%	30/30	5	0.0%
	100	3	0/30	7200	50.7%	11.1%	30/30	12	0.0%
		4	0/30	7200	51.5%	12.2%	30/30	11	0.0%
P	50	2	22/30	3782	2.8%	0.1%	30/30	0	0.0%
		3	13/30	5615	8.3%	0.3%	30/30	0	0.0%
	100	2	0/30	7200	57.6%	15.3%	30/30	57	0.0%
		3	0/30	7200	52.3%	11.2%	30/30	45	0.0%

Table 1: Results for the MILP formulations $[F1]$ and $[F2]$ according to the instance characteristics

latter case, the gap is very high. It seems to come from a very weak upper bound resulting from solving the linear relaxation of model $[F1]$ and, to a lesser extent, from a difficulty for finding high-quality primal solutions (cf. column Gap$^{[F2]}$). We conclude that time discretization allows us to build a model that is more interesting from a computational point of view.

We empirically tested the performance of solving $[F2]$ for larger instances. We report the results in Table 2. The solver returns an optimal solution for all instances of set A, whereas it fails to provide an optimal solution for instances of set P within the time limit for most of the instances with 150 jobs or more. The gap for the unsolved instances of set P can still be important after two hours of computational time.

Our results suggest that instances for which the profit associated with each job is correlated with the processing time on the main resource (i.e., instances of set P) are the most difficult. These instances have jobs with a shorter duration than instances of set A and more jobs can be scheduled. Using Proposition 2, we are able for the instances of set A to detect a significant number of jobs that are dominated by other jobs (a job j dominates a job j' if it is only profitable to schedule j' when j is scheduled). We can also detect some jobs (around 5% on average) that are never profitable to schedule. We also observe that instances with the smallest number of secondary resources or a large number of tasks are more challenging.

Set	n	m	#Opt	Time	Gap
A	150	3	30/30	18	0.0%
		4	30/30	17	0.0%
	200	3	30/30	23	0.0%
		4	30/30	25	0.0%
	250	3	30/30	28	0.0%
		4	30/30	29	0.0%
	300	3	30/30	35	0.0%
		4	30/30	36	0.0%
	400	3	30/30	48	0.0%
		4	30/30	49	0.0%
	500	3	30/30	59	0.0%
		4	30/30	63	0.0%

Set	n	m	#Opt	Time	Gap
P	150	2	0/30	7200	2.3%
		3	24/30	3367	0.1%
	200	2	0/30	7200	3.8%
		3	1/30	6995	0.9%
	250	2	0/30	7200	4.1%
		3	2/30	6986	0.9%

Table 2: Additional results for the MILP formulation $[F2]$ according to the instance characteristics

8.2 Results for the heuristic algorithm

For these experiments, we considered all the instances and performed 10 runs for each instance. We ran the heuristic algorithm with the following parameters set after performing preliminary experiments: $\delta^{max} = 200000$, $\theta^{max} = 2000$, $\epsilon = 0.02$, $\beta^{min} = 4$, $\beta^{max} = \max\{4, 0.2|\mathcal{J}^{sch}|\}$, $\alpha = 0.25$, $\eta = 2$, $\gamma = 5$.

For comparison purposes, we have computed the best known solution (BKS) for each instance from the results of Maschler and Raidl (2018), Horn et al. (2021a), our heuristic algorithm, the MILP formulation $[F2]$

and the BCP algorithm applied to solve $[F3]$ (the two latter methods are not considered for instances of set P with more than 250 jobs). As a result, all BKS for set A and almost all BKS for instances of set P with less than 250 jobs are optimal.

Table 3 reports the average over 10 runs of the CPU time in seconds (Time) and the average gap (Gap) with respect to the BKSs. We compute the gap with respect to a value $\bar{z}$ as $(\bar{z} - z)/\bar{z}$, where z is the mean (Mean), minimum (Worst) or maximum (Best) objective function value of the best solution computed by the heuristic over the 10 runs. We report results after 20000 and 200000 iterations. To judge the efficiency of the heuristic to find good-quality solutions in a short computational time, we also tested solving formulation $[F2]$ in a heuristic way. Specifically, for each instance we set a time limit equal to the average computational time of the heuristic for 200000 iterations. We report the gap with respect to the BKSs ($[F2]$ – Heur.). We also report in the two last columns the gap with respect to the BKS for the GVNS introduced by Maschler and Raidl (2018) (GVNS) and the TDC with A* search and cost-based filtering described in Horn et al. (2021a) (TDC-A*-f). These two last methods were run for 900 seconds.

First, if we look at the results for 20000 iterations, we can see that the heuristic usually finds very quickly good solutions, especially for the instances of set A. Nonetheless, we see that the remaining iterations are necessary to find better solutions. The computational time remains reasonably low. The heuristic exhibits a stable behavior for instances of set A, while the difference between the best and the worst solution found over the 10 runs is on average 0.5% for instances of set P. On the tested instances, results are slightly better when m is larger. We clearly see that our heuristic outperforms existing methods in terms of solution quality. It computes a solution with an objective strictly better than or identical to the best reported in the literature for all but one of the 960 instances of sets A and P if we consider the best result over the 10 runs for each instance (and all but 61 out of the 960 instances if we consider the worst result). The improvement increases with the number of jobs and are primarily for instances of set P. We also observe that the results found by the MILP solver with model $[F2]$ are better for the instances of set A. It finds the optimal solutions in less than one minute on average and clearly outperforms our heuristic. For the instances of set P, the solver is not competitive for finding good primal solutions for instances with 150 jobs and more.

To have a better overview, Figure 4 shows the number of instances whose computed solution has a gap with respect to the BKS smaller than a certain value. As a whole, we see that our heuristic outperforms the other methods.

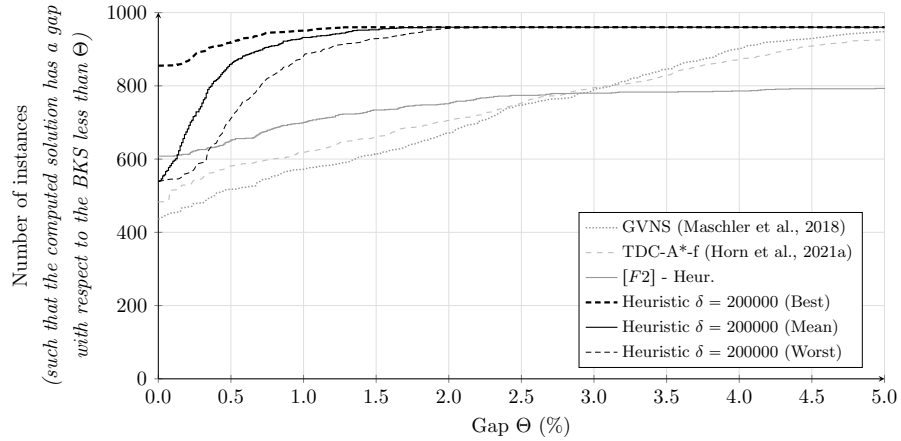


Figure 4: Performance profiles of several heuristic methods on the 960 instances of set A and set P

We tested the LNS without the ILS framework (it corresponds to setting $\theta^{max} = \delta^{max}$). Results are worse on average. We observe a 0.5% difference between the average gap with respect to the BKSs with and without ILS for the instances of set P with 200 and 400 jobs.

We finally tested removing the random perturbation performed before calling the repair operator. We observe worst (or almost equal) results on average for the instances of set A and for the instances of set P with a number of jobs less than or equal to 200. For the instances of set P with a number of jobs larger than 200

		Heuristic / $\delta = 20000$				Heuristic / $\delta = 200000$				[$F2$] -	Literature	
		Gap			Time	Gap			Time	Heur.	GVNS	TDC-A*-f
Set	n	Worst	Mean	Best		Worst	Mean	Best		Gap	Gap	Gap
A	50	0.16%	0.03%	0.00%	2	0.01%	0.00%	0.00%	22	0.00%	0.00%	0.03%
	100	0.13%	0.03%	0.00%	7	0.00%	0.00%	0.00%	68	0.00%	0.00%	0.02%
	150	0.22%	0.05%	0.00%	12	0.01%	0.00%	0.00%	116	0.00%	0.04%	0.03%
	200	0.15%	0.05%	0.00%	16	0.03%	0.01%	0.00%	162	0.00%	0.06%	0.08%
	250	0.10%	0.04%	0.00%	21	0.01%	0.00%	0.00%	209	0.00%	0.10%	0.12%
	300	0.15%	0.06%	0.01%	25	0.02%	0.01%	0.00%	254	0.00%	0.23%	0.17%
	400	0.22%	0.06%	0.00%	34	0.03%	0.01%	0.00%	337	0.00%	0.37%	0.09%
	500	0.24%	0.07%	0.01%	42	0.06%	0.02%	0.01%	421	0.00%	0.26%	0.12%
	All	0.17%	0.05%	0.00%	20	0.02%	0.01%	0.00%	199	0.00%	0.13%	0.08%
P	50	0.36%	0.06%	0.00%	2	0.00%	0.00%	0.00%	16	0.00%	0.04%	0.00%
	100	1.11%	0.42%	0.02%	6	0.28%	0.07%	0.00%	60	0.02%	0.71%	0.29%
	150	1.69%	1.05%	0.47%	13	0.76%	0.38%	0.13%	129	0.61%	1.92%	1.31%
	200	2.09%	1.54%	1.04%	23	1.27%	0.86%	0.48%	235	1.19%	2.76%	2.59%
	250	1.70%	1.24%	0.75%	36	0.91%	0.59%	0.25%	367	2.81%	2.73%	2.61%
	300	1.36%	0.91%	0.48%	51	0.61%	0.31%	0.01%	517	8.16%	2.57%	2.90%
	400	1.23%	0.81%	0.40%	84	0.49%	0.26%	0.00%	860	14.64%	3.48%	3.94%
	500	1.18%	0.77%	0.40%	125	0.48%	0.24%	0.00%	1282	17.86%	4.21%	4.36%
	All	1.33%	0.84%	0.44%	42	0.60%	0.34%	0.11%	433	5.66%	2.30%	2.25%
Set	m											
A	3	0.18%	0.05%	0.01%	20	0.02%	0.01%	0.00%	196	0.00%	0.11%	0.10%
	4	0.16%	0.05%	0.00%	20	0.02%	0.01%	0.00%	201	0.00%	0.15%	0.07%
P	2	1.54%	1.02%	0.56%	47	0.74%	0.43%	0.14%	483	6.17%	2.59%	1.89%
	3	1.11%	0.67%	0.32%	38	0.46%	0.25%	0.08%	383	5.16%	2.02%	2.61%

Table 3: Comparison of the results obtained by different heuristic methods

instances, we observe slightly better results with a difference between the average gap with respect to the BKS approximately equal to 0.05% for the solutions computed by the heuristic with and without this perturbation.

8.3 Results for solving [F3] with a BCP algorithm

We tested the performance of solving [$F3$] with our BCP algorithm on the instances with up to 250 jobs (set P) and up to 500 jobs (set A). We set a two-hour CPU time limit (single thread). We used as input the value of the best solution found by our heuristic algorithm wiht $\delta = 200000$ described in Section 7 minus one (the total computational time we report includes the time to compute this solution). We tested the following BCP configurations:

- **G**: pricing problem defined on graph G
- **G + R1Cs**: configuration **G** + separation of R1Cs and use of the limited memory version of these cuts
- **G′**: pricing problem defined on graph G'
- **G′ + R1Cs**: configuration **G′** + separation of R1Cs and use of the limited memory version of these cuts
- **G′ + R1Cs + AR**: configuration **G′ + R1Cs** + reduction of the number of arcs in G' according to the dominance rules introduced in §3.

Table 4 gives the comparison of four BCP configurations. It shows the average values for the root gap (Root Gap), the root CPU time (Root Time), the gap for the computed solution with respect to the best dual bound (Gap), the number of branch-and-bound nodes (TN), the pricing problem solution time (SP Time), and the total CPU time (Time). It also reports the number of instances optimally solved within the time limit (#Opt). We compute the root gap considering the best (primal) solution found before starting branching and the linear relaxation value. For unsolved instances, the computational time is set to the time limit.

Solutions to instances of set P contain a lot more jobs than those of set A, which explained why they are easier to solve with a BCP algorithm. We see that results obtained for the BCP configurations with a

	G							**G + R1Cs**						
Set	Root Gap	Root Time	Gap	#Opt	TN	SP Time	Time	Root Gap	Root Time	Gap	#Opt	TN	SP Time	Time
A	0.10%	19	0.01%	475/480	26	55	320	0.09%	20	0.00%	480/480	1	17	219
P	1.05%	79	0.53%	180/300	71	1745	3253	0.77%	656	0.39%	188/300	7	2511	2968
	G′							**G′ + R1Cs**						
Set	Root Gap	Root Time	Gap	#Opt	TN	SP Time	Time	Root Gap	Root Time	Gap	#Opt	TN	SP Time	Time
A	0.09%	289	0.00%	478/480	9	267	514	0.09%	292	0.00%	480/480	1	254	493
P	1.01%	244	0.43%	197/300	49	2058	2931	0.74%	993	0.23%	230/300	4	1913	2289

Table 4: Results for the BCP configurations according to the pricing problem definition and the use of R1Cs

pricing problem defined on G are better than for those with a pricing problem defined on G' for instances of set A, whereas it is the opposite for instances of set P (with a greater difference). When using graph G, dominance performed at node $v_\star$ reduces the total number of generated labels, which reduces the pricing problem solution time. However, for instances of set P, we remove a larger proportion of (bucket) arcs with the reduced cost-based filtering procedures when using G'. Moreover, the set of schedules considered with graph G' is smaller than the one considered with graph G. Indeed, it is forbidden by construction of G' to have the same job using the main resource more than once in a row in a particular time window. This does not seem to have a significant impact on the root gap for the tested instances, but this makes finding integer feasible solution easier when starting branching. When using R1Cs with graph G, we observe that the number of non-dominated labels when solving the pricing problem explodes. This can be explained by larger arc memory sets than with graph G'. This also confirms that solutions to the pricing problem contain more jobs with multiple occurrences for BCP configurations **G** than for BCP configurations **G′**. Our results also show that using R1Cs improves the convergence of the BCP algorithm as it is able to compute more proven optimal solutions within the same computational time. The root gap decreases by approximately 0.3% for instances of set P and the branch-and-bound tree is smaller.

Table 5 shows a comparison of the results obtained when reducing the number of arcs in G'. Although we only see a slight improvement in the computational time (due to a faster resolution of the pricing problem), the arc reduction procedure nevertheless allows to solve to optimality four additional instances.

	G′ + R1Cs							**G′ + R1Cs + AR**						
Set	Root Gap	Root Time	Gap	#Opt	TN	SP Time	Time	Root Gap	Root Time	Gap	#Opt	TN	SP Time	Time
A	0.09%	292	0.00%	480/480	1	254	493	0.09%	170	0.00%	480/480	1	151	370
P	0.74%	993	0.23%	230/300	4	1913	2289	0.74%	907	0.22%	234/300	4	1831	2204

Table 5: Results for the BCP algorithm depending on using the arc reduction results

Table 6 compares solving [$F3$] with our BCP algorithm to solving model [$F2$] with a MILP solver. Our BCP configurations are not competitive to solve the easiest instances of set A. For the more difficult and more challenging instances of set P, they outperform the MILP solver, which demonstrates their interest. Instances with more than 200 jobs are challenging to solve as there are still a lot of them currently open. Instances with more secondary resources are also more difficult to solve by the BCP algorithm, which is explained by weaker dominance rules.

		G + R1Cs			**G′ + R1Cs + AR**			[$F2$]		
Set	n	Opt	Gap	Time	Opt	Gap	Time	Opt	Gap	Time
A	50	60/60	0.00%	24	60/60	0.00%	24	60/60	0.00%	6
	100	60/60	0.00%	71	60/60	0.00%	78	60/60	0.00%	12
	150	60/60	0.00%	122	60/60	0.00%	139	60/60	0.00%	18
	200	60/60	0.00%	173	60/60	0.00%	206	60/60	0.00%	24
	250	60/60	0.00%	222	60/60	0.00%	293	60/60	0.00%	29
	300	60/60	0.00%	278	60/60	0.00%	406	60/60	0.00%	36
	400	60/60	0.00%	382	60/60	0.00%	701	60/60	0.00%	49
	500	60/60	0.00%	483	60/60	0.00%	1109	60/60	0.00%	61
	All	480/480	0.00%	219	480/480	0.00%	370	480/480	0.00%	29
P	50	60/60	0.00%	16	60/60	0.00%	16	60/60	0.00%	0
	100	60/60	0.00%	63	60/60	0.00%	64	60/60	0.00%	51
	150	58/60	0.02%	577	60/60	0.00%	280	24/60	1.20%	5284
	200	10/60	0.69%	6662	44/60	0.20%	3405	1/60	2.35%	7097
	250	0/60	1.24%	7522	10/60	0.90%	7253	2/60	2.50%	7093
	All	188/300	0.39%	2968	234/300	0.22%	2204	147/300	1.21%	3267
Set	m									
A	3	240/240	0.00%	209	240/240	0.00%	273	240/240	0.00%	29
	4	240/240	0.00%	230	240/240	0.00%	467	240/240	0.00%	29
P	2	97/150	0.46%	2819	130/150	0.18%	1726	60/150	2.04%	4331
	3	91/150	0.32%	3117	104/150	0.26%	2681	87/150	0.38%	3479

Table 6: Results for the best BCP configurations and for formulation [$F2$] depending on the instance characteristics

Last, to provide a global overview and finer comparison of solution time, Figure 5 shows the number of instances optimally solved according to the solution time for the different exact methods presented in this paper. We come to the same conclusion as before. The MILP solver performs better for instances of set A and that the BCP configuration **G+R1Cs+AR** performs better for instances of set P.

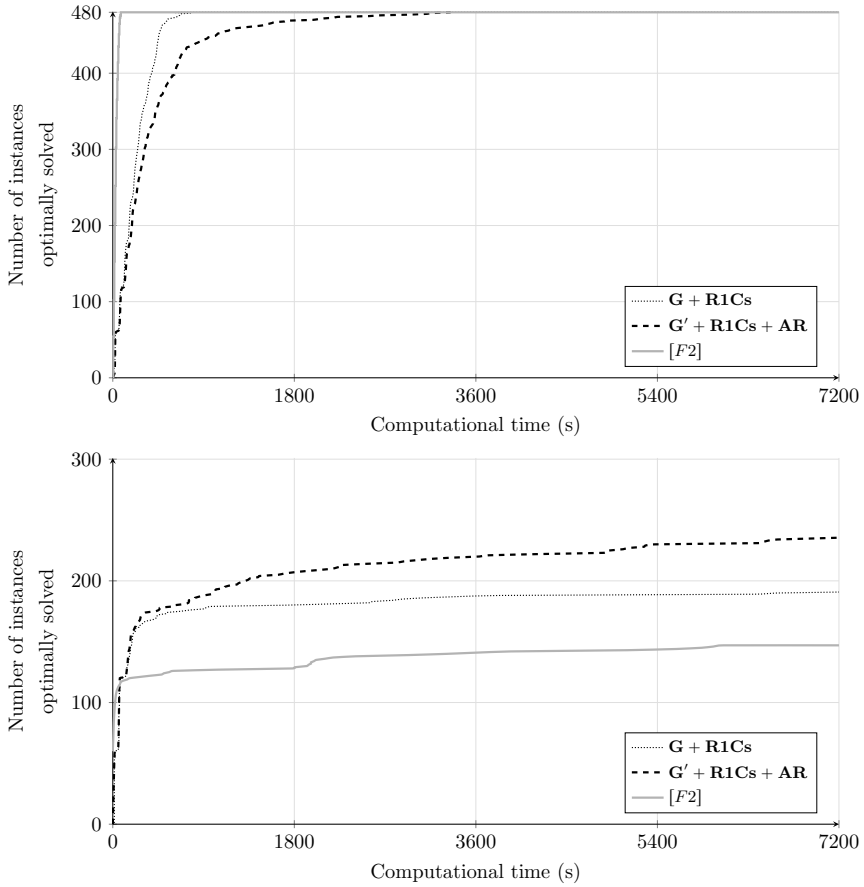


Figure 5: Performance charts of the best exact algorithms on the instances of set A (top) and set P (bottom)

9 Conclusion

We have proposed a new pseudo-polynomial time-indexed formulation ([$F2$]) for the PC-JSOCMSR. Computational experiments have shown that a MILP solver finds significantly better solutions with this formulation in comparison with an existing compact order-based formulation.

As an exact solution method, we have introduced a BCP algorithm to solve an extended MILP formulation of the problem based on the generation of schedules that may contain several occurrences of the same job. We have presented two alternative ways of defining the pricing problem as a resource-constrained shortest path problem. We have also presented dominance rules to reduce the search space. While it only has a slight impact on the efficiency of the method, these rules are general enough to be interesting for other algorithms. Our BCP algorithm is able to compute proven optimal solutions for more instances in comparison with solving [$F2$] with a MILP solver. Using R1Cs with limited memory to strengthen the linear relaxation provides better results. We have optimally solved instances with up to 250 jobs (500 jobs on a particular set of instances), while in the literature only a few instances with 50 jobs were optimally solved.

We have developed an ILS to heuristically solve the PC-JSOCMSR. Our experiments have shown this heuristic algorithm gives equal or better results than existing heuristic methods of the literature for more than 93% of the tested instances in the worst case of our 10 runs (99% in the best case) and the gap between the solution value returned by our heuristic and that of the literature is on average for the most difficult set of instances between approximately 1.5% and 2%.

As a perspective, refining relaxations of the time-indexed formulation [$F2$] may be a path worth investigating for solving exactly larger instances. A possible extension of this work concerns the case of multiple main resources with every secondary resource being associated with a dedicated main resource (e.g., the case of patients that

can choose to have their cancer treatment in one location out of several).

Acknowledgments

The experiments presented in this paper were carried out using the Federative Platform for Research in Computer Science and Mathematics (PlaFRIM), created under the Inria PlaFRIM development action with support from Institut Polytechnique de Bordeaux, Laboratoire Bordelais de Recherche en Informatique, Institut de Mathématiques de Bordeaux, Conseil Régional d'Aquitaine, Université de Bordeaux, Centre National de la Recherche Scientifique, and Agence Nationale de la Recherche in accordance with the "Programme d'Investissements d'Avenir" (see `www.plafrim.fr/en/home`).

References

Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, 246(2):345–378.

Castro, M. P., Cire, A. A., and Beck, J. C. (2022). Decision diagrams for discrete optimization: A survey of recent advances. arXiv preprint arXiv:2201.11536, `https://arxiv.org/abs/2201.11536`.

Dror, M. (1994). Note on the complexity of the shortest path models for column generation in vrptw. *Operations Research*, 42(5):977–978.

Gunawan, A., Lau, H. C., and Vansteenwegen, P. (2016). Orienteering Problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 255(2):315–332.

Horn, M., Maschler, J., Raidl, G. R., and Rönnberg, E. (2021a). A*-based construction of decision diagrams for a prize-collecting scheduling problem. *Computers and Operations Research*, 126:105125.

Horn, M., Raidl, G. R., and Rönnberg, E. (2018). An A* algorithm for solving a prize-collecting sequencing problem with one common and multiple secondary resources and time windows. In *Proceedings of PATAT*, pages 235–256.

Horn, M., Raidl, G. R., and Rönnberg, E. (2021b). A* search for prize-collecting job sequencing with one common and multiple secondary resources. *Annals of Operations Research*, 302:477–505.

Irnich, S., Desaulniers, G., Desrosiers, J., and Hadjar, A. (2010). Path-Reduced Costs for Eliminating Arcs in Routing and Scheduling. *INFORMS Journal on Computing*, 22(2):297–313.

Koulamas, C. and Kyparisis, G. J. (2008). Single-machine scheduling problems with past-sequence-dependent setup times. *European Journal of Operational Research*, 187(3):1045–1049.

Maschler, J. and Raidl, G. R. (2018). Multivalued decision diagrams for a prize-collecting sequencing problem. In *Proceedings of the 12th International Conference of the Practice and Theory of Automated Timetabling, PATAT 2018. pp. 375–397.*

Maschler, J. and Raidl, G. R. (2020). Particle therapy patient scheduling with limited starting time variations of daily treatments. *International Transactions in Operational Research*, 27(1):458–479.

Pecin, D., Contardo, C., Desaulniers, G., and Uchoa, E. (2017a). New Enhancements for the Exact Solution of the Vehicle Routing Problem with Time Windows. *INFORMS Journal on Computing*, 29(3):489–502.

Pecin, D., Pessoa, A., Poggi, M., and Uchoa, E. (2017b). Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, 9(1):61–100.

Pessoa, A., Sadykov, R., and Uchoa, E. (2021). Solving bin packing problems using vrpsolver models. *Operations Research Forum*, 2(2):20.

Pessoa, A., Sadykov, R., Uchoa, E., and Vanderbeck, F. (2018). Automation and combination of linear-programming based stabilization techniques in column generation. *INFORMS Journal on Computing*, 30(2):339–360.

Sadykov, R., Uchoa, E., and Pessoa, A. (2021). A bucket graph–based labeling algorithm with application to vehicle routing. *Transportation Science*, 55(1):4–28.

Sadykov, R. and Vanderbeck, F. (2021). BaPCod - a generic branch-and-price code. Technical report, Inria Bordeaux Sud-Ouest.

van der Veen, J. A. A., Woeginger, G. J., and Zhang, S. (1998). Sequencing jobs that require common resources on a single machine: A solvable case of the TSP. *Mathematical Programming*, 82(1-2):235–254.

Vansteenwegen, P., Souffriau, W., and Oudheusden, D. V. (2011). The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1–10.

Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., and Van Oudheusden, D. (2009). Iterated local search for the team orienteering problem with time windows. *Computers & Operations Research*, 36(12):3281–3290.

S.1 Proofs

S.1.1 Proof of Proposition 1

Proof. Let $j, j' \in \mathcal{J}$ such that $j \neq j'$, $k \in [\![1, w_j]\!]$, and $k' \in [\![1, w_{j'}]\!]$, and assume that $t^1_{jk} \leqslant t^1_{j'k'}$, $t^2_{jk} \leqslant t^2_{j'k'}$, $p^-_j \geqslant p^-_{j'}$, and $p^+_j \leqslant p^+_{j'}$.

Let $\mathsf{S}_{j' \to j}$ be a schedule with job j' using the main resource just before job j uses it. Let $start_j$ and $start_{j'}$ be the starting time of job j and j' in this schedule. As $t^1_{jk} \leqslant t^1_{j'k'}$, we can always assume that $start_j = start_{j'} + p_{j'}$. In $\mathsf{S}_{j' \to j}$, the main resource is either available or used by the two jobs from time $start_{j'} + p^-_{j'}$ to time $start_j + p^-_j + p^0_j$. The secondary resource r_j is used by the two jobs from time $start_{j'}$ to time $start_j + p_j$.

Let $\mathsf{S}_{j \to j'}$ be the schedule obtained from $\mathsf{S}_{j' \to j}$ by switching the processing order of job j and j' and with j starting at time $start_{j'}$ and j' starting at time $start_{j'} + p_j$. As $t^1_{jk} \leqslant t^1_{j'k'}$, job j can start at time $start_{j'}$. As $t^2_{jk} \leqslant t^2_{j'k'}$, job j' can start at time $start_{j'} + p_j$. This means that in $\mathsf{S}_{j \to j'}$ the secondary resource r_j is still used by the two jobs from time $start_{j'}$ to time $start_{j'} + p_j + p_{j'} = start_j + p_j$. In $\mathsf{S}_{j \to j'}$, the main resource is used from time $start_{j'} + p^-_j$ to time $start_j + p^-_{j'} + p^0_{j'}$. As $p^-_j \geqslant p^-_{j'}$, we have $start_{j'} + p^-_j \geqslant start_{j'} + p^-_{j'}$. As $p^+_j \leqslant p^+_{j'}$, we also have $start_j + p^-_{j'} + p^0_{j'} \leqslant start_{j'} + p_j + p^-_{j'} + p^0_{j'}$. This means that the main resource is required in $\mathsf{S}_{j \to j'}$ during a sub-interval of the interval during which it is required in $\mathsf{S}_{j' \to j}$.

We can therefore always convert a schedule in which j' uses the main resource just before j to a schedule in which j uses the main resource just before j'. □

S.1.2 Proof of Proposition 2

We first introduce the functions ϕ^{es}_{jk} and ϕ^{ls}_{jk} defined for every $j \in \mathcal{J}$ and $k \in [\![1, w_j]\!]$:

- $\phi^{es}_{jk}(t, t')$ denotes the earliest starting time possible for a job j in time window T_{jk} if the resources 0 and r_j become available at time t and t' (∞ if not existing):

$$\phi^{es}_{jk}(t, t') = \begin{cases} t^1_{jk} & \text{if } \max\{t - p^-_j, t'\} \leqslant t^1_{jk} \\ \max\{t - p^-_j, t'\} & \text{if } t^1_{jk} < \max\{t - p^-_j, t'\} \leqslant t^2_{jk} - p_j \\ \infty & \text{if } t^2_{jk} - p_j < \max\{t - p^-_j, t'\} \end{cases}$$

- $\phi^{ls}_{jk}(t, t')$ denotes the latest starting time possible for a job j in time window T_{jk} if the resources 0 and r_j, after their use by j, should become available at time t and t' ($-\infty$ if not existing):

$$\phi^{ls}_{jk}(t, t') = \begin{cases} t^2_{jk} - p_j & \text{if } \min\{t - p^-_j - p^0_j, t' - p_j\} \geqslant t^2_{jk} - p_j \\ \max\{t - p^-_j, t'\} & \text{if } t^1_{jk} \leqslant \min\{t - p^-_j - p^0_j, t' - p_j\} < t^2_{jk} - p_j \\ -\infty & \text{if } \min\{t - p^-_j - p^0_j, t' - p_j\} < t^1_{jk} \end{cases}$$

From the functions ϕ^{es}_{jk} and ϕ^{es}_j defined for every $j \in \mathcal{J}$ and $k \in [\![1, w_j]\!]$, we mathematically define the functions ϕ^{es}_j and ϕ^{ls}_j as follows:

$$\phi^{es}_j(t, t') = \min_{k \in [\![1, w_j]\!]} \left\{ \phi^{es}_{jk}(t, t') \right\}$$

$$\phi^{ls}_j(t, t') = \max_{k \in [\![1, w_j]\!]} \left\{ \phi^{ls}_{jk}(t, t') \right\}$$

We then mathematically translate the content of Proposition 2.

- Let $j' \in \mathcal{J}$, $k' \in [\![1, w_{j'}]\!]$, and $j \in \mathcal{J} \backslash \{j'\}$. Job j cannot use the main resource before job j' scheduled during time window $T_{j'k'}$ if:

$$\begin{cases} t^1_{j1} + p_j > t^2_{j'k'} - p_{j'} & \text{if } r_j = r_{j'} \\ t^1_{j1} + p^-_j + p^0_j > t^2_{j'k'} - p_{j'} + p^-_{j'} & \text{if } r_j \neq r_{j'} \end{cases} \tag{26}$$

Conditions (26) state that scheduling j at its earliest starting time implies that j' cannot be scheduled during time window $T_{j'k'}$.

- Let $j' \in \mathcal{J}$, $k' \in [\![1, w_{j'}]\!]$, $j'' \in \mathcal{J}\backslash\{j', j''\}$, $k'' \in [\![1, w_{j''}]\!]$, and $j \in \mathcal{J}\backslash\{j','' \}$. Job j cannot use the main resource before this resource is used by job j' scheduled during time window $T_{j'k'}$ and then by job j'' scheduled during time window $T_{j''k''}$ if:

$$\begin{cases} t^1_{j1} + p_j > \phi^{ls}_{j'k'}(t^2_{j''k''} - p_{j''} + p^-_{j''}, t^2_{j''k''} - p_{j''}) & \text{if } r_j = r_{j'} = r_{j''} \\ t^1_{j1} + p_j > \phi^{ls}_{j'k'}(t^2_{j''k''} - p_{j''} + p^-_{j''}, t^2) & \text{if } r_j = r_{j'} \wedge r_{j''} \neq r_j \\ t^1_{j1} + p^-_j + p^0_j > \phi^{ls}_{j'k'}(t^2_{j''k''} - p_{j''} + p^-_{j''}, t^2) + p^-_{j'} \vee t^1_{j1} + p_j > t^2_{j''k''} - p_{j''} & \text{if } r_j = r_{j''} \wedge r_{j'} \neq r_j \\ t^1_{j1} + p^-_j + p^0_j > \phi^{ls}_{j'k'}(t^2_{j''k''} - p_{j''} + p^-_{j''}, t^2_{j''k''} - p_{j''}) + p^-_{j'} & \text{if } r_j \neq r_{j'} \wedge r_{j'} = r_{j''} \\ t^1_{j1} + p^-_j + p^0_j > \phi^{ls}_{j'k'}(t^2_{j''k''} - p_{j''} + p^-_{j''}, t^2) + p^-_{j'} & \text{if } r_j \neq r_{j'} \wedge r_j \neq r_{j''} \wedge r_{j'} \neq r_{j''} \end{cases} \tag{27}$$

Conditions (27) state that scheduling j at its earliest starting time implies that j' and j'' cannot use the main resource in this order after j.

- Let $j' \in \mathcal{J}$, $k' \in [\![1, w_{j'}]\!]$, and $j \in \mathcal{J}\backslash\{j'\}$. Job j cannot use the main resource after job j' scheduled during time window $T_{j'k'}$ if:

$$\begin{cases} t^1_{j'k'} + p_{j'} > t^2_{jw_j} - p_j & \text{if } r_j = r_{j'} \\ t^1_{j'k'} + p^-_{j'} + p^0_{j'} > t^2_{jw_j} - p_j + p^-_j & \text{if } r_j \neq r_{j'} \end{cases} \tag{28}$$

Conditions (28) state that scheduling j at its latest starting time implies that j' cannot be scheduled during time window $T_{j'k'}$.

- Let $j' \in \mathcal{J}$, $k' \in [\![1, w_{j'}]\!]$, $j'' \in \mathcal{J}\backslash\{j'\}$, $k'' \in [\![1, w_{j''}]\!]$, and $j \in \mathcal{J}\backslash\{j', j''\}$. Job j cannot use the main resource after this resource is used by job j' scheduled during time window $T_{j'k'}$ and then by job j'' scheduled during time window $T_{j''k''}$ if:

$$\begin{cases} \phi^{es}_{j''k''}(t^1_{j'k'} + p^-_{j'} + p^+_{j'}, t^1_{j'k'} + p_{j'}) + p_{j''} > t^2_{jw_j} - p_j & \text{if } r_j = r_{j'} = r_{j''} \\ \phi^{es}_{j''k''}(t^1_{j'k'} + p^-_{j'} + p^+_{j'}, 0) + p^-_{j''} + p^0_{j''} > t^2_{jw_j} - p_j + p^-_j \vee t^1_{j'k'} + p_{j'} > t^2_{jw_j} - p_j & \text{if } r_j = r_{j'} \wedge r_{j''} \neq r_j \\ \phi^{es}_{j''k''}(t^1_{j'k'} + p^-_{j'} + p^+_{j'}, 0) + p_{j''} > t^2_{jw_j} - p_j & \text{if } r_j = r_{j''} \wedge r_{j'} \neq r_j \\ \phi^{es}_{j''k''}(t^1_{j'k'} + p^-_{j'} + p^+_{j'}, t^1_{j'k'} + p_{j'}) + p^-_{j''} + p^0_{j''} > t^2_{jw_j} - p_j + p^-_j & \text{if } r_j \neq r_{j'} \wedge r_{j'} = r_{j''} \\ \phi^{es}_{j''k''}(t^1_{j'k'} + p^-_{j'} + p^+_{j'}, 0) + p^-_{j''} + p^0_{j''} > t^2_{jw_j} - p_j + p^-_j & \text{if } r_j \neq r_{j'} \wedge r_j \neq r_{j''} \wedge r_{j'} \neq r_{j''} \end{cases} \tag{29}$$

Conditions (29) state that scheduling j at its latest starting time implies that j' and j'' cannot use the main resource in this order before j.

- Let $j' \in \mathcal{J}$, $k' \in [\![1, w_{j'}]\!]$, and $j \in \mathcal{J}\backslash\{j'\}$. Job j can use the main resource just before job j' scheduled during $T_{j'k'}$ without preventing j' to start its execution at time $t^1_{j'k'}$ and imposing restriction on the processing of subsequent jobs:

$$\begin{cases} t^1_{j1} + p_j \leqslant t^1_{j'k'} & \text{if } r_{j'} = r_j \\ t^1_{j1} + p^-_j + p^0_j \leqslant t^1_{j'k'} + p^-_{j'} \wedge t^1_{j1} + p_j \leqslant t^1_{j'k'} + p^-_{j'} + p^0_{j'} - \hat{p}^-_{r_j} & \text{if } r_{j'} \neq r_j \end{cases} \tag{30}$$

Conditions (30) state that there exists a starting time for j such that j' can start at any time during time window $T_{j'k'}$ and that the next job using the secondary resource r_j (possibly j') can start at any time. If $r_j = r_{j'}$, only the secondary resource needs consideration. If $r_j \neq r_{j'}$, observe that the main resource (after its use by j') can never be available for subsequent jobs before time $t^1_{j'k'} + p^-_{j'} + p^0_{j'}$. The earliest possible starting time of a task j''' such that $r_{j'''} = r_j$ is therefore equal to $t^1_{j'k'} + p^-_{j'} + p^0_{j'} - p^-_{j'''} \geqslant t^1_{j'k'} + p^-_{j'} + p^0_{j'} - \hat{p}^-_{r_j}$.

- Let $j' \in \mathcal{J}$, $k' \in [\![1, w_{j'}]\!]$, and $j \in \mathcal{J}\backslash\{j'\}$. Job j can use the main resource just after job j' scheduled during $T_{j'k'}$ without preventing j' to start its execution at time $t^2_{j'k'} - p_{j'}$ and imposing restriction on the

processing of precedent jobs if:

$$\begin{cases} t^2_{j'k'} + p_j \leqslant t^2_{jw_j} & \text{if } r_{j'} = r_j \\ t^2_{j'k'} - p^+_{j'} + p^0_j + p^+_j \leqslant t^2_{jw_j} \wedge t^2_{j'k'} - p_{j'} + p^-_{j'} + \hat{p}^+_{r_j} + p_j \leqslant t^2_{jw_j} & \text{if } r_{j'} \neq r_j \end{cases} \tag{31}$$

Conditions (31) state that there exists a starting time for j such that j' can start at any time during time window $T_{j'k'}$ and that the previous job using the secondary resource r_j (possibly j') can start at any time. As before, only the secondary resource needs consideration if $r_j = r_{j'}$. If $r_j \neq r_{j'}$, observe that the main resource (before its use by j') should be available not later than time $t^2_{j'k'} - p_{j'} + p^-_j$. The latest possible completion time of a task j''' such that $r_{j'''} = r_j$ is therefore equal to $t^2_{j'k'} - p_{j'} + p^-_{j'} + p^+_{j'''} \leqslant t^2_{j'k'} - p_{j'} + p^-_{j'} + \hat{p}^+_{r_j}$.

- Let $j' \in \mathcal{J}$, $k' \in [\![1, w_{j'}]\!]$, $j'' \in \mathcal{J}\backslash\{j'\}$, $k'' \in [\![1, w_{j''}]\!]$, and $j \in \mathcal{J}\backslash\{j', j''\}$. Job j can use the main resource just after j' and just before j'' without preventing j'' to start its execution at time $t^1_{j''k''}$ and j' to end at time $t^2_{j'k'}$ and imposing restriction on the processing of precedent and subsequent jobs if:

$$\begin{cases} \phi^{es}_j(t^2_{j'k'} - p^+_{j'}, t^2_{j'k'}) + p_j \leqslant t^1_{j''k''} & \text{if } r_j = r_{j''} = r_{j'} \\ \phi^{es}_j(t^2_{j'k'} - p^+_{j'}, t^2_{j'k'} - p^+_{j'} - p^0_{j'} + \hat{p}^+_{r_j}) + p_j \leqslant t^1_{j''k''} & \text{if } r_j = r_{j''} \wedge r_j \neq r_{j'} \\ \left(\phi^{es}_j(t^2_{j'k'} - p^+_{j'}, t^2_{j'k'}) + p^-_j + p^0_j \leqslant t^1_{j''k''} + p^-_{j''}\right) & \\ \wedge \left(\phi^{es}_j(t^2_{j'k'} - p^+_{j'}, t^2_{j'k'}) + p_j \leqslant t^1_{j''k''} + p^-_{j''} + p^0_{j''} - \hat{p}^-_{r_j}\right) & \text{if } r_j = r_{j'} \wedge r_j \neq r_{j''} \\ \left(\phi^{es}_j(t^2_{j'k'} - p^+_{j'}, t^2_{j'k'} - p^+_{j'} - p^0_{j'} + \hat{p}^+_{r_j}) + p^-_j + p^0_j \leqslant t^1_{j''k''} + p^-_{j''}\right) & \\ \wedge \left(\phi^{es}_j(t^2_{j'k'} - p^+_{j'}, t^2_{j'k'} - p^+_{j'} - p^0_{j'} + \hat{p}^+_{r_j}) + p_j \leqslant t^1_{j''k''} + p^-_{j''} + p^0_{j''} - \hat{p}^-_{r_j}\right) & \text{if } r_j \notin \{r_{j'}, r_{j''}\} \end{cases} \tag{32}$$

Conditions (32) state that there exists a starting time for j such that j' and j'' can start at any time during time window $T_{j'k'}$ and $T_{j''k''}$ and such that the previous and next job using the secondary resource r_j (possibly j' and j'') can start at any time. We use the same arguments as above to derive the sufficient condition. Since j is scheduled between j' and j'', the use of function $\phi^{es}_j(.,.)$ is required.

In Lemma 1, we identify when a job j can always be scheduled during time window T_{jk} in the place of a job j' scheduled during time window $T_{j'k'}$.

Lemma 1. *Let $j, j' \in \mathcal{J}$ such that $j \neq j' \wedge r_j = r_{j'}$, $k \in [\![1, w_j]\!]$, and $k' \in [\![1, w_{j'}]\!]$. The following conditions are sufficient to ensure that job j can always be scheduled during time window T_{jk} in the place of job j' scheduled during time window $T_{j'k'}$.*

$$\exists d \in [\max\{0, p^-_{j'} - p^-_j\}, \min\{p^-_{j'} - p^-_j + p^0_{j'} - p^0_j, p_{j'} - p_j\}] : t^1_{jk} \leqslant t^1_{j'k'} + d \ \wedge \ t^2_{jk} + p_{j'} - p_j - d \geqslant t^2_{j'k'} \tag{33}$$

Proof. Let $\mathsf{S}_{j'}$ be a schedule with job j' and $start_{j'}$ the starting time of this job. In $\mathsf{S}_{j'}$, the main resource is used by j' from time $start_{j'} + p^-_{j'}$ to time $start_{j'} + p^-_{j'} + p^0_{j'}$. The secondary resource r_j is used by j' from time $start_{j'}$ to time $start_{j'} + p_{j'}$.

Let S_j be the schedule obtained from $\mathsf{S}_{j'}$ by replacing j' by j and setting $start_j = start_{j'} + d$. As $start_{j'} \geqslant t^1_{j'k'}$ and $t^1_{jk} \leqslant t^1_{j'k'} + d$, we have $start_j \geqslant t^1_{jk}$. As $start_{j'} + p_{j'} \leqslant t^2_{j'k'}$ and $t^2_{jk} + p_{j'} - p_j - d \geqslant t^2_{j'k'}$, we have $start_j + p_j \leqslant t^2_{jk}$.

In S_j, the main resource is used from time $start_j + p^-_j$ to time $start_j + p^-_j + p^0_j$. Since $d \geqslant p^-_{j'} - p^-_j$, we have $start_j + p^-_j \geqslant start_{j'} + p^-_{j'}$. Since $d \leqslant p^-_{j'} - p^-_j + p^0_{j'} - p^0_j$, we have $start_j + p^-_j + p^0_j \leqslant start_{j'} + p^-_{j'} + p^0_{j'}$. This means that the main resource is required in S_j during a sub-interval of the interval during which it is required in $\mathsf{S}_{j'}$.

In S_j, the secondary resource is used from time $start_j$ to time $start_j + p_j$. Since $d \geqslant 0$, we have $start_j \geqslant start_{j'}$. Since $d \leqslant p_{j'} - p_j$, we have $start_j + p_j \leqslant start_{j'} + p_{j'}$. This means that the secondary resource is required in S_j during a sub-interval of the interval during which it is required in $\mathsf{S}_{j'}$.

We can therefore always replace job j' scheduled in time window $T_{j'k'}$ by job j. □

We finally reformulate Proposition 2 using the conditions previously introduced.

Proposition 2. *Assume that a job $j' \in \mathcal{J}$ is scheduled during time window $T_{j'k'}$ with $k' \in [\![1, w_{j'}]\!]$. There exists an optimal schedule in which:*

1- Job j' does not use the main resource just after its use by another job $j'' \in \mathcal{J}\backslash\{j'\}$ during time window $T_{j''k''}$ with $k'' \in [\![1, w_{j''}]\!]$ (resp. j' is not the first job using the main resource) if there exists a job $j \in \mathcal{J}\backslash\{j', j''\}$ (resp. $j \in \mathcal{J}\backslash\{j'\}$) such that (j'', k'', j', k', j) satisfies (27) and (29)(resp. (j', k', j) satisfies (28)) and that satisfies at least one of the two following conditions:

a) $r_j = r_{j'}$, $b_j \geqslant b_{j'}$ and $\exists k \in [\![1, w_j]\!]$ such that (j, j', k, k') satisfies (33)

b) (j'', k'', j) satisfies (31) and (j', k', j) satisfies (30) (resp. (j', k', j) satisfies (30))

2- Job j' does not use the main resource just before its use by another job $j'' \in \mathcal{J}\backslash\{j'\}$ during time window $T_{j''k''}$ with $k'' \in [\![1, w_{j''}]\!]$ (resp. j' is not the last job using the main resource) if there exists a job $j \in \mathcal{J}\backslash\{j', j''\}$ (resp. $j \in \mathcal{J}\backslash\{j'\}$) such that (j', k', j'', k'', j) satisfies (27) and (29)(resp. (j', k', j) satisfies (26)) and that satisfies at least one of the two following conditions:

a) $r_j = r_{j'}$, $b_j \geqslant b_{j'}$ and $\exists k \in [\![1, w_j]\!]$ such that (j, j', k, k') satisfies (33)

b) (j', k', j) satisfies (31) and (j'', k'', j) satisfies (30) (resp. (j', k', j) satisfies (31))

Proof. Case 1: Let S be a schedule where the main resource is consecutively used by j'' and j' (in that order). This means that j is not scheduled in S.

a) It is always possible to build a more profitable schedule S' that is identical to S with the exception of either job j scheduled such that it uses the main resource between j'' and j' (resp. before j'), or job j scheduled in the place of j'.

b) It is always possible to build a more profitable schedule S' that is identical to S with the exception of job j scheduled such that it uses the main resource between j'' and j' (resp. before j')

Case 2: Let now S be a schedule where the main resource is consecutively used by j' and j'' (in that order). This means that j is not scheduled in S.

a) It is always possible to build a more profitable schedule S' that is identical to S with the exception of either job j scheduled such that it uses the main resource between j' and j'' (resp. after j'), or job j scheduled in the place of j'.

b) It is always possible to build a more profitable schedule S' that is identical to S with the exception of job j scheduled such that it uses the main resource between j' and j'' (resp. after j')

□

S.2 Labeling algorithms

Algorithm 3: `runMonodirectionalAlgorithm(dir)`

input : `dir` $\in$ {`forward`, `backward`}
output: a set of extended non-dominated labels
if `dir` = `forward` **then** **else**
$L \leftarrow$ `createInitialLabel`(`dir`)
$N \leftarrow \{L\}$, $E \leftarrow \varnothing$
while $N \neq \varnothing$ **do**
  $L \leftarrow$ `getNextLabelToExtend`$(N,$ `dir`$)$ ▷ See Function 6
  $N \leftarrow N \backslash \{L\}$
  for $L' \in E$ such that $v^{L'} = v^L$ and $Q_0^{L'} < Q_0^L$ **do**
    if `Dominates`$(L', L,$ `dir`$)$ **then** **go to** line 3 ▷ See Function 8
  end
  $E \leftarrow E \cup \{L\}$
  if `dir` = `forward` **then**
    $\Gamma \leftarrow \Gamma^+(v^L, Q_0^L)$ ▷ Set of outgoing arcs from v^L at time Q_0^L in the bucket implementation of the graph
  else
    $\Gamma \leftarrow \Gamma^-(v^L, Q_0^L)$ ▷ Set of incoming arcs at v^L at time Q_0^L in the bucket implementation of the graph
  end
  for $a \in \Gamma$ **do**
    if `Extend`$(L, a, L',$ `dir`$)$ **and not** `IsBorderCrossed`$(L',$ `dir`$)$ **then** ▷ See Functions 7 and 9
      for $L'' \in N \cup E$ such that $v^{L''} = v^{L'}$ and $Q_0^{L''} = Q_0^{L'}$ **do**
        if `Dominates`$(L'', L',$ `dir`$)$ **then** **continue** the loop of line 3 with the next arc
      end
      for $L'' \in N$ such that $v^{L''} = v^{L'}$ and $Q_0^{L''} = Q_0^{L'}$ **do**
        if `Dominates`$(L', L'',$ `dir`$)$ **then** $N \leftarrow N \backslash \{L''\}$
      end
      $N \leftarrow N \cup \{L'\}$
    end
  end
end
return E

Algorithm 4: `runBidirectionalAlgorithm()`

output: the best solution and its value
$\vec{E} \leftarrow$ `runMonodirectionalAlgorithm(forward)`
$\overleftarrow{E} \leftarrow$ `runMonodirectionalAlgorithm(backward)`
$\mathsf{S}^* \leftarrow \varnothing, c^* \leftarrow 0$
for $\vec{L} \in \vec{E}$ **do**
  for $a \in \Gamma^+(v^{\vec{L}})$ **do**
    for $\overleftarrow{L} \in \overleftarrow{E}$ such that $v^{\overleftarrow{L}}$ is the head of a **do**
      $c \leftarrow$ `ConcatenationCost`$(\vec{L}, a, \overleftarrow{L})$ ▷ See Function 10
      if $c > c^*$ **then**
        $\mathsf{S}^* \leftarrow$ `getSchedule`$(\vec{L}, a, \overleftarrow{L})$ ▷ Compute the semi-active schedule (result of the concatenation)
        $c^* \leftarrow c$
      end
    end
  end
end
return S^*, c^*

Function 5: `CreateInitialLabel`

input : `dir` $\in$ {`forward`, `backward`}
output: the initial label
Function `CreateInitialLabel(dir)`:
  if `dir` = `forward` **then**
    return $(0, v_{source}, (l_{v_{source},r})_r, \varnothing)$ ▷ Only for the algorithm of §5.2
    return $(0, v_{source}, (l_{v_{source},r})_r, (0)_g)$ ▷ Only for the algorithm of §6.2
  else
    return $(0, v_{sink}, (u_{v_{sink},r})_r, \varnothing)$ ▷ Only for the algorithm of §5.2
    return $(0, v_{sink}, (u_{v_{sink},r})_r, (0)_g)$ ▷ Only for the algorithm of §6.2
  end

Function 6: `getNextLabelToExtend`

input : a set of non-extended labels N, `dir` $\in$ {`forward`, `backward`}
output: the label to extend
Function `getNextLabelToExtend`$(N,$ `dir`$)$:
  if `dir` = `forward` **then**
    return the first label $\vec{L}$ of N in non-decreasing order of the vector $Q_r^{\vec{L}}$ with two labels of N being compared using the lexicographical order on the Cartesian product $\mathbb{Z}^{m+1}$
  else
    return the first label $\overleftarrow{L}$ of N in non-increasing order of the vector $Q_r^{\overleftarrow{L}}$ with two labels of N being compared using the lexicographical order on the Cartesian product $\mathbb{Z}^{m+1}$
  end

Function 7: `Extend`

input : L is a label, a is an arc, `dir` $\in \{$`forward`, `backward`$\}$, L' is the result of the extension of L along a in direction `dir`

output: `true` if the extension succeeds, `false` otherwise

Function `Extend`$(L, a, L', $`dir`$)$**:**

if `dir` = `forward` **then**

$v^{L'} \leftarrow$ head of a

$\Delta \leftarrow \max\left\{0,\ l_{v^{L'},0} - Q_0^L - q_{a,0}\ ,\ \max_{r\in\mathcal{R}}\left\{Q_r^L + q_{a,r} - u_{v^{L'},r}\right\}\right\}$

$Q_0^{L'} \leftarrow Q_0^L + q_{a,r} + \Delta$

if $Q_0^{L'} > u_{v^{L'},0}$ **then return** `false`

for $r \in \mathcal{R}$ **do**

$Q_r^{L'} \leftarrow \max\left\{l_{a,r}\ ,\ Q_r^L + q_{a,r} - \Delta\right\}$

end

else

$v^{L'} \leftarrow$ queue of a

$\Delta \leftarrow \max\left\{0,\ Q_0^L - q_{a,0} - u_{v^{L'},0}\ ,\ \max_{r\in\mathcal{R}}\left\{l_{v^{L'},r} - (Q_r^L - q_{a,r})\right\}\right\}$

$Q_0^{L'} \leftarrow Q_0^L - q_{a,r} - \Delta$

if $Q_0^{L'} < l_{v^{L'},0}$ **then return** `false`

for $r \in \mathcal{R}$ **do**

$Q_r^{L'} \leftarrow \min\left\{u_{v^{L'},r}\ ,\ Q_r^L - q_{a,r} + \Delta\right\}$

end

end

if $v^{L'} \in \bigcup_{j\in J^{\bar{L}}} V_j$ **then return** `false`

$c^{L'} \leftarrow c^L + c_a$ ▷ Only for the algorithm of §5.2

if $\exists j \in \mathcal{J}$ such that $v^{L'} \in V_j$ **then** $J^{L'} \leftarrow J^L \cup \{j\}$ **else** $J^{L'} \leftarrow J^L$ ▷ Only for the algorithm of §5.2

$\bar{c}^{L'} \leftarrow \bar{c}^L + \bar{c}_a$ ▷ Only for the algorithm of §6.2

for $g \in \mathcal{G}$ **do** ▷ Only for the algorithm of §6.2

if $a \notin AM(g)$ **then** $E_g^{L'} \leftarrow E_g^L$ **else** $E_g^{L'} \leftarrow 0$

if $v \in V_{j(g)}$ **then**

$E_g^{L'} \leftarrow E_g^{L'} + 0.5$

if $E_g^{L'} = 1$ **then** $E_g^{L'} \leftarrow 0$, $\bar{c}^{L'} \leftarrow \bar{c}^{L'} - \sigma_g$

end

end

return `true`

Function 8: `Dominates`

input : L and L' are two labels, `dir` $\in$ {`forward`, `backward`}

output: `true` if L dominates L', `false` otherwise

Function `Dominates`$(L, L', \texttt{dir})$**:**

- **if** $v^L \neq v^{L'}$ **then return** `false`
- **if** $J^L \nsubseteq J^{L'}$ **then return** `false` ▷ Only for the algorithm of §5.2
- **if** $c^L < c^{L'}$ **then return** `false` ▷ Only for the algorithm of §5.2
- **if** $\bar{c}^L < \bar{c}^{L'} + \sum_{g \in \mathcal{G}: E_g^L > E_g^{L'}} \sigma_g$ **then return** `false` ▷ Only for the algorithm of §6.2
- **if** `dir` = `forward` **then**
  - **if** $Q_0^L > Q_0^{L'}$ **then return** `false`
  - **for** $r \in \mathcal{R}$ **do**
    - **if** $Q_r^L - (Q_0^{L'} - Q_0^L) > Q_r^{L'}$ **then return** `false`
  - **end**
- **else**
  - **if** $Q_0^L < Q_0^{L'}$ **then return** `false`
  - **for** $r \in \mathcal{R}$ **do**
    - **if** $Q_r^L + (Q_0^L - Q_0^{L'}) < Q_r^{L'}$ **then return** `false`
  - **end**
- **end**
- **return** `true`

Function 9: `IsBorderCrossed`

input : L is a label, `dir` $\in$ {`forward`, `backward`}

output: `true` if the border in direction `dir` is crossed by L, `false` otherwise

Function `IsBorderCrossed`$(L, \texttt{dir})$**:**

- **if** `dir` = `forward` **then**
  - **return** $Q_0^L > t^*$
- **else**
  - **return** $Q_0^L \leqslant t^*$
- **end**

Function 10: `ConcatenationCost`

input : $\vec{L}$ is a forward label, a an arc, and $\overleftarrow{L}$ a backward label

output: $-\infty$ if the two labels cannot be concatenated,
the cost of the solution resulting from the concatenation otherwise

Function `ConcatenationCost`$(\vec{L}, a, \overleftarrow{L})$**:**

- **if** $Q_0^{\vec{L}} + q_{a,0} > Q_0^{\overleftarrow{L}}$ **then return** $-\infty$
- **for** $r \in \mathcal{R}$ **do**
  - **if** $Q_r^{\vec{L}} + q_{a,r} - \left(Q_0^{\overleftarrow{L}} - (Q_0^{\vec{L}} + q_{a,0})\right) > Q_r^{\overleftarrow{L}}$ **then return** $-\infty$
- **end**
- **if** $J^{\vec{L}} \cap J^{\overleftarrow{L}} \neq \varnothing$ **then return** $-\infty$
- **return** $c^{\vec{L}} + c_a + c^{\overleftarrow{L}}$ ▷ Only for the algorithm of §5.2
- **return** $\bar{c}^{\vec{L}} + \bar{c}_a + \bar{c}^{\overleftarrow{L}} - \sum_{g \in \mathcal{G}: E_g^{\vec{L}} + E_g^{\overleftarrow{L}} \geqslant 1} \sigma_g$ ▷ Only for the algorithm of §6.2

S.3 Example of the bucket graph implementation

In the forward (resp. backward) bucket graph implementation, the arcs incoming to v_{sink} (resp. going out of v_{source}) are represented as dotted arcs to make the graphs easier to read.

A label L', obtained by extension from label L along an arc $a = ((v,t),(v',t'))$ of the bucket graph implementation of G (or G') is not necessarily contained in the bucket (v',t'). It is contained in the bucket $(v',Q_0^{L'})$.

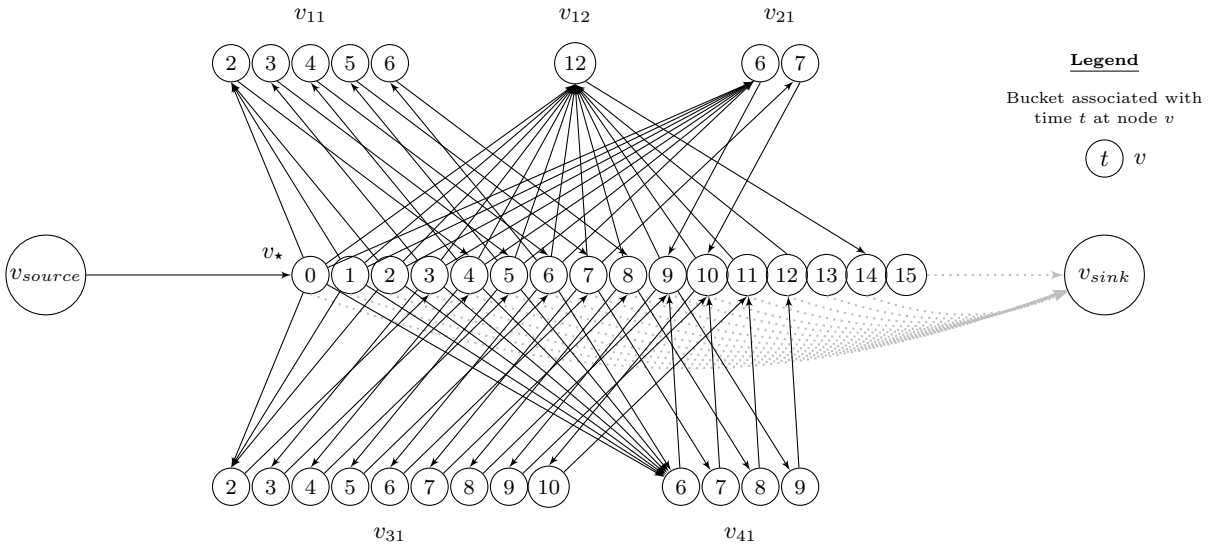


Figure 6: The forward bucket graph implementation of the graph G displayed in Figure 1

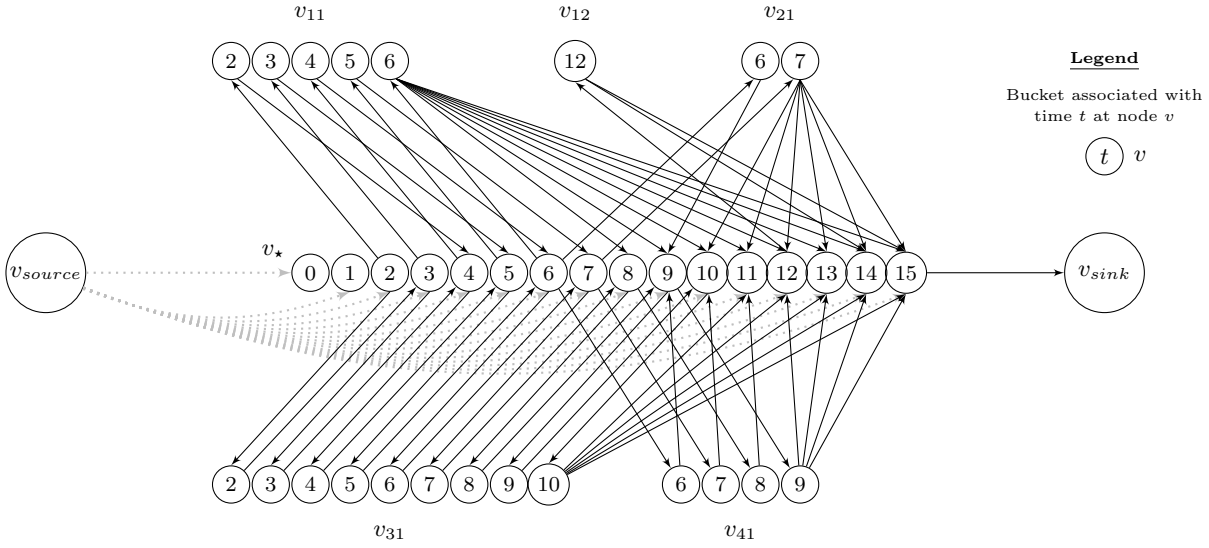


Figure 7: The backward bucket graph implementation of the graph G displayed in Figure 2

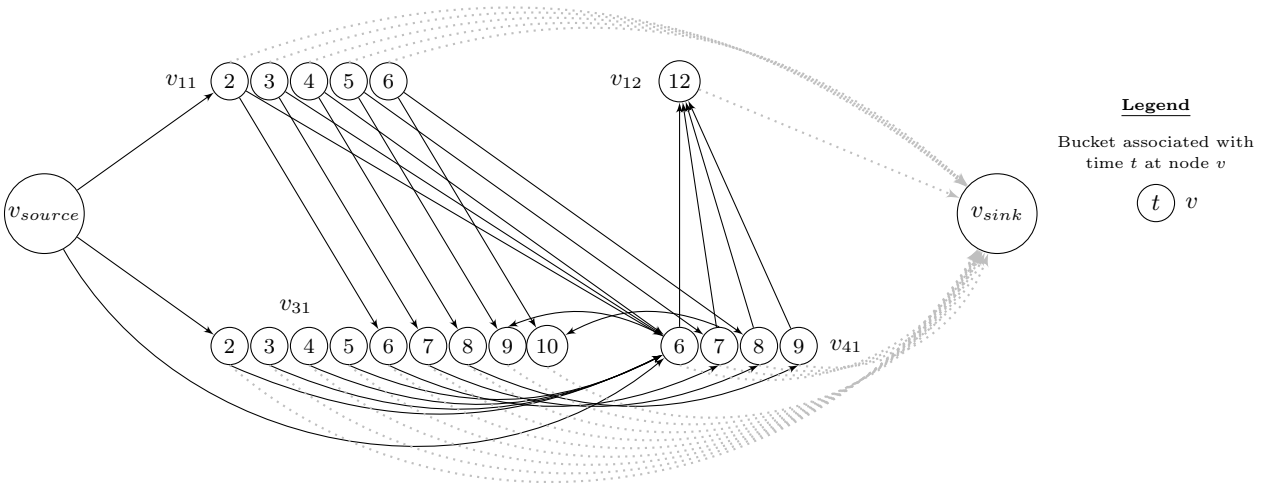


Figure 8: The forward bucket graph implementation of G' displayed in Figure 3

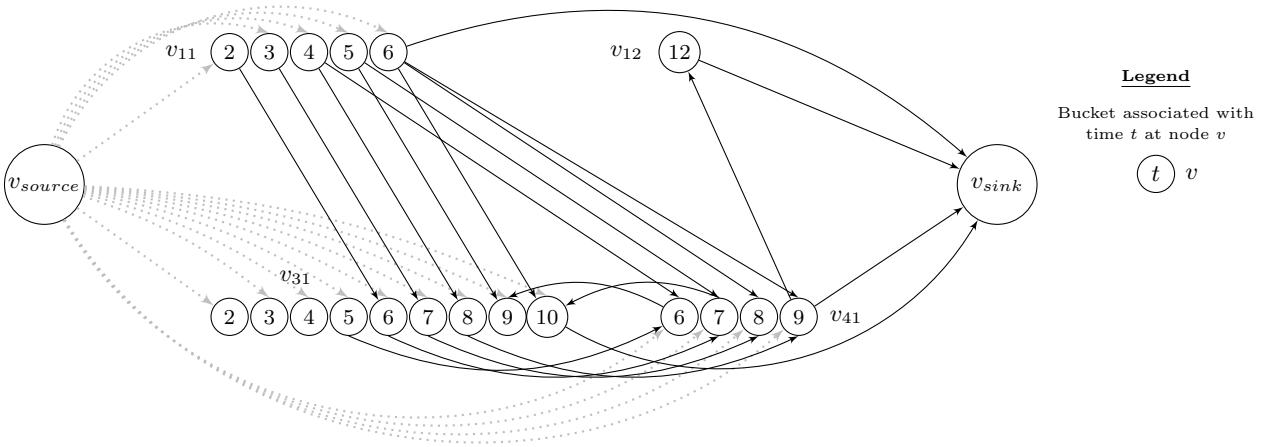


Figure 9: The backward bucket graph implementation of G' displayed in Figure 3

S.4 Detailed computational results

			Heuristic / $\delta = 20000$				Heuristic / $\delta = 200000$				[$F2$] -	Literature	
			Gap			Time	Gap			Time	Heur.	GVNS	TDC-A*-f
Set	n	m	Worst	Mean	Best		Worst	Mean	Best		Gap	Gap	Gap
A	50	3	0.24%	0.04%	0.00%	2	0.00%	0.00%	0.00%	21	0.00%	0.00%	0.00%
		4	0.08%	0.03%	0.00%	2	0.01%	0.00%	0.00%	22	0.00%	0.00%	0.06%
	100	3	0.14%	0.03%	0.00%	7	0.01%	0.00%	0.00%	66	0.00%	0.00%	0.03%
		4	0.11%	0.03%	0.00%	7	0.00%	0.00%	0.00%	70	0.00%	0.00%	0.01%
	150	3	0.16%	0.05%	0.00%	11	0.01%	0.00%	0.00%	113	0.00%	0.05%	0.06%
		4	0.27%	0.05%	0.01%	12	0.02%	0.01%	0.00%	119	0.00%	0.03%	0.01%
	200	3	0.20%	0.07%	0.00%	16	0.04%	0.01%	0.00%	161	0.00%	0.07%	0.10%
		4	0.10%	0.04%	0.00%	16	0.03%	0.01%	0.00%	163	0.00%	0.05%	0.05%
	250	3	0.12%	0.04%	0.00%	21	0.00%	0.00%	0.00%	206	0.00%	0.03%	0.08%
		4	0.08%	0.03%	0.00%	21	0.02%	0.00%	0.00%	211	0.00%	0.16%	0.17%
	300	3	0.13%	0.06%	0.03%	25	0.04%	0.01%	0.00%	254	0.00%	0.26%	0.25%
		4	0.16%	0.05%	0.00%	25	0.01%	0.00%	0.00%	255	0.00%	0.20%	0.08%
	400	3	0.22%	0.04%	0.00%	33	0.02%	0.00%	0.00%	333	0.00%	0.33%	0.13%
		4	0.21%	0.07%	0.00%	34	0.04%	0.01%	0.00%	342	0.00%	0.42%	0.05%
	500	3	0.21%	0.06%	0.02%	41	0.07%	0.02%	0.01%	416	0.00%	0.15%	0.12%
		4	0.27%	0.08%	0.01%	42	0.04%	0.01%	0.01%	426	0.00%	0.36%	0.13%
P	50	2	0.27%	0.05%	0.00%	2	0.00%	0.00%	0.00%	16	0.00%	0.03%	0.00%
		3	0.43%	0.07%	0.00%	2	0.00%	0.00%	0.00%	17	0.00%	0.05%	0.00%
	100	2	0.97%	0.35%	0.00%	6	0.22%	0.06%	0.00%	59	0.03%	0.59%	0.24%
		3	1.20%	0.48%	0.03%	6	0.33%	0.08%	0.00%	61	0.01%	0.83%	0.33%
	150	2	1.83%	1.10%	0.46%	13	0.86%	0.40%	0.12%	132	0.86%	2.13%	0.99%
		3	1.51%	0.96%	0.46%	13	0.64%	0.36%	0.13%	126	0.36%	1.72%	1.63%
	200	2	2.52%	1.90%	1.32%	25	1.55%	1.07%	0.59%	253	1.57%	3.11%	2.59%
		3	1.61%	1.16%	0.75%	21	0.96%	0.64%	0.35%	218	0.81%	2.41%	2.60%
	250	2	2.18%	1.65%	1.07%	40	1.24%	0.83%	0.37%	405	3.33%	3.26%	2.31%
		3	1.20%	0.82%	0.42%	32	0.58%	0.35%	0.12%	328	2.29%	2.20%	2.90%
	300	2	1.67%	1.18%	0.63%	57	0.79%	0.42%	0.01%	587	11.30%	3.07%	2.17%
		3	1.04%	0.64%	0.32%	44	0.42%	0.21%	0.00%	448	5.01%	2.07%	3.62%
	400	2	1.51%	1.03%	0.52%	95	0.63%	0.34%	0.00%	975	14.69%	3.88%	3.29%
		3	0.95%	0.60%	0.28%	73	0.35%	0.18%	0.00%	744	14.59%	3.08%	4.59%
	500	2	1.35%	0.91%	0.50%	139	0.59%	0.30%	0.00%	1439	17.54%	4.62%	3.55%
		3	0.98%	0.62%	0.30%	110	0.37%	0.18%	0.00%	1125	18.18%	3.80%	5.17%

Table 7: Detailed computational results for the heuristic methods

Set	n	m	Root Gap	Root Time	Gap	#Opt	TN	SP Time	Time
A	50	3	0.15%	2	0.03%	29	91	105	263
		4	0.09%	1	0.00%	30	1	1	24
	100	3	0.11%	3	0.03%	29	83	100	308
		4	0.08%	4	0.00%	30	2	3	74
	150	3	0.12%	5	0.00%	30	6	10	135
		4	0.09%	7	0.01%	29	48	116	366
	200	3	0.09%	7	0.00%	30	54	74	400
		4	0.12%	14	0.00%	30	41	82	351
	250	3	0.08%	7	0.00%	30	1	5	214
		4	0.08%	19	0.00%	30	1	17	230
	300	3	0.08%	12	0.00%	30	2	17	276
		4	0.08%	30	0.00%	30	1	27	286
	400	3	0.09%	20	0.01%	29	38	90	611
		4	0.08%	51	0.00%	30	1	47	393
	500	3	0.09%	21	0.00%	30	1	16	437
		4	0.09%	96	0.01%	29	41	170	757
P	50	2	0.83%	0	0.00%	30	1	0	16
		3	0.72%	0	0.00%	30	1	0	17
	100	2	0.44%	2	0.00%	30	1	1	61
		3	0.47%	5	0.00%	30	1	3	65
	150	2	1.27%	35	0.03%	29	88	368	1185
		3	0.75%	63	0.08%	28	60	571	1292
	200	2	1.84%	64	1.52%	1	268	2991	7356
		3	0.99%	187	0.70%	2	116	4443	7213
	250	2	2.19%	112	2.04%	0	125	3459	7610
		3	1.02%	319	0.93%	0	46	5612	7717

Table 8: Detailed computational results for BCP configuration **G**

Set	n	m	Root Gap	Root Time	Gap	#Opt	TN	SP Time	Time
A	50	3	0.13%	3	0.00%	30	1	1	24
		4	0.09%	1	0.00%	30	1	1	24
	100	3	0.11%	4	0.00%	30	1	2	69
		4	0.08%	4	0.00%	30	1	3	73
	150	3	0.08%	5	0.00%	30	1	3	118
		4	0.08%	7	0.00%	30	1	6	127
	200	3	0.09%	8	0.00%	30	1	5	168
		4	0.10%	14	0.00%	30	1	9	177
	250	3	0.08%	7	0.00%	30	1	5	214
		4	0.08%	19	0.00%	30	1	17	230
	300	3	0.08%	16	0.00%	30	1	12	270
		4	0.08%	30	0.00%	30	1	27	286
	400	3	0.08%	29	0.00%	30	1	24	370
		4	0.08%	51	0.00%	30	1	47	393
	500	3	0.09%	21	0.00%	30	1	16	437
		4	0.09%	103	0.00%	30	1	91	528
P	50	2	0.83%	0	0.00%	30	1	0	16
		3	0.72%	0	0.00%	30	1	0	17
	100	2	0.38%	2	0.00%	30	1	1	61
		3	0.35%	4	0.00%	30	1	2	65
	150	2	0.37%	69	0.00%	30	1	29	202
		3	0.40%	345	0.04%	28	3	725	952
	200	2	1.06%	1297	0.71%	7	16	5301	6177
		3	0.88%	1424	0.67%	3	16	6293	7147
	250	2	1.70%	1328	1.57%	0	21	6474	7639
		3	0.97%	2087	0.91%	0	14	6289	7405

Table 9: Detailed computational results for BCP configuration **G + R1Cs**

Set	n	m	Root Gap	Root Time	Gap	#Opt	TN	SP Time	Time
A	50	3	0.15%	4	0.03%	29	119	117	264
		4	0.09%	3	0.00%	30	1	2	26
	100	3	0.08%	12	0.00%	30	1	9	78
		4	0.08%	16	0.00%	30	1	13	86
	150	3	0.08%	31	0.00%	30	1	24	144
		4	0.08%	47	0.00%	30	1	39	166
	200	3	0.08%	61	0.00%	30	1	48	222
		4	0.11%	94	0.00%	30	2	90	280
	250	3	0.08%	89	0.00%	30	1	68	295
		4	0.08%	211	0.00%	30	1	180	422
	300	3	0.08%	157	0.00%	30	1	127	410
		4	0.08%	410	0.00%	30	1	360	665
	400	3	0.08%	275	0.00%	30	1	220	607
		4	0.08%	843	0.00%	30	1	746	1185
	500	3	0.09%	485	0.00%	30	1	397	900
		4	0.09%	1896	0.01%	29	6	1840	2471
P	50	2	0.86%	0	0.00%	30	1	0	16
		3	0.72%	0	0.00%	30	1	0	17
	100	2	0.39%	4	0.00%	30	1	3	63
		3	0.42%	8	0.00%	30	1	7	69
	150	2	1.09%	42	0.00%	30	13	75	263
		3	0.72%	96	0.00%	30	35	374	760
	200	2	1.79%	167	0.70%	14	244	3252	5876
		3	0.98%	494	0.71%	3	80	5231	7079
	250	2	2.15%	456	1.97%	0	86	5338	7610
		3	1.02%	1169	0.94%	0	27	6298	7562

Table 10: Detailed computational results for BCP configuration $\mathbf{G}'$

Set	n	m	Root Gap	Root Time	Gap	#Opt	TN	SP Time	Time
A	50	3	0.15%	5	0.00%	30	1	3	27
		4	0.09%	3	0.00%	30	1	2	26
	100	3	0.08%	12	0.00%	30	1	9	78
		4	0.08%	16	0.00%	30	1	13	86
	150	3	0.08%	31	0.00%	30	1	24	144
		4	0.08%	47	0.00%	30	1	39	166
	200	3	0.08%	60	0.00%	30	1	47	221
		4	0.11%	107	0.00%	30	1	97	281
	250	3	0.08%	89	0.00%	30	1	68	295
		4	0.08%	210	0.00%	30	1	179	421
	300	3	0.08%	157	0.00%	30	1	126	411
		4	0.08%	409	0.00%	30	1	359	665
	400	3	0.08%	270	0.00%	30	1	214	602
		4	0.08%	847	0.00%	30	1	748	1189
	500	3	0.09%	491	0.00%	30	1	399	907
		4	0.09%	1918	0.00%	30	1	1744	2365
P	50	2	0.86%	0	0.00%	30	1	0	16
		3	0.72%	0	0.00%	30	1	0	17
	100	2	0.39%	4	0.00%	30	1	3	63
		3	0.35%	8	0.00%	30	1	6	69
	150	2	0.37%	52	0.00%	30	1	33	184
		3	0.42%	185	0.00%	30	2	178	364
	200	2	0.98%	828	0.01%	29	7	1304	1838
		3	0.84%	1724	0.32%	15	9	4928	5621
	250	2	1.53%	2522	1.05%	5	9	6082	7145
		3	0.97%	4611	0.88%	1	5	6591	7570

Table 11: Detailed computational results for BCP configuration $\mathbf{G'} + \mathbf{R1Cs}$

Set	n	m	Root Gap	Root Time	Gap	#Opt	TN	SP Time	Time
A	50	3	0.13%	3	0.00%	30	1	2	24
		4	0.09%	2	0.00%	30	1	1	25
	100	3	0.08%	7	0.00%	30	1	5	73
		4	0.08%	13	0.00%	30	1	10	82
	150	3	0.08%	16	0.00%	30	1	12	129
		4	0.08%	29	0.00%	30	1	24	148
	200	3	0.08%	31	0.00%	30	1	23	192
		4	0.08%	57	0.00%	30	1	48	221
	250	3	0.08%	42	0.00%	30	1	32	249
		4	0.08%	126	0.00%	30	1	110	338
	300	3	0.08%	73	0.00%	30	1	57	326
		4	0.08%	231	0.00%	30	1	206	486
	400	3	0.08%	206	0.00%	30	1	192	558
		4	0.08%	502	0.00%	30	1	456	844
	500	3	0.09%	214	0.00%	30	1	171	630
		4	0.09%	1161	0.00%	30	1	1073	1588
P	50	2	0.86%	0	0.00%	30	1	0	16
		3	0.72%	0	0.00%	30	1	0	17
	100	2	0.39%	2	0.00%	30	1	1	61
		3	0.35%	5	0.00%	30	1	4	66
	150	2	0.38%	39	0.00%	30	1	19	171
		3	0.41%	151	0.00%	30	3	197	388
	200	2	0.98%	687	0.00%	30	6	1067	1588
		3	0.84%	1554	0.39%	14	8	4546	5223
	250	2	1.53%	2181	0.92%	10	9	5752	6794
		3	0.97%	4450	0.89%	0	5	6723	7712

Table 12: Detailed computational results for BCP configuration $\mathbf{G'} + \mathbf{R1Cs} + \mathbf{AR}$