# A Decision Diagram Approach for the Parallel Machine Scheduling Problem with Chance Constraints

**Nicolás Casassus** · **Margarita Castro** · **Gustavo Angulo**

**Abstract** The Chance-Constrained Parallel Machine Scheduling Problem (CC-PMSP) assigns jobs with uncertain processing times to machines, ensuring that each machine's availability constraints are met with a certain probability. We present a decomposition approach where the master problem assigns jobs to machines, and the subproblems schedule the jobs on each machine while verifying the solution's feasibility under the chance constraint. We propose two different Decision Diagram (DD) formulations to solve the subproblems and generate cuts. The first formulation employs DDs with a linear cost function, while the second uses a non-linear cost function to reduce the diagram's size. We show how to generate no-good and irreducible infeasible subsystem (IIS) cuts based on our DDs. Additionally, we extend the cuts proposed by Lozano & Smith [42] to solve two-stage stochastic programming models. Our DD-based methodology outperforms traditional integer programming (IP) models designed to solve the CC-PMSP in several instances. Specifically, our best DD-based approach solves 55 more instances than the best IP alternative (from a total of 405) and typically achieves smaller gaps (50% vs. 120% gap on average).

N. Casassus
Department of Industrial and Systems Engineering, Pontificia Universidad Católica de Chile

M. Castro
Department of Industrial and Systems Engineering, Pontificia Universidad Católica de Chile
E-mail: margarita.castro@uc.cl

G. Angulo
Department of Industrial and Systems Engineering, Pontificia Universidad Católica de Chile

## 1 INTRODUCTION

Scheduling problems represent a fundamental area within operational research, with numerous practical applications across economic and industrial sectors [48]. These problems focus on efficiently allocating limited resources over time to execute a set of tasks while optimizing one or more objectives. They naturally arise in various contexts, such as manufacturing environments where jobs must be assigned to machines and their processing order determined, hospitals scheduling surgeries and medical interventions, airports establishing takeoff and landing sequences, and logistics centers coordinating loading, unloading, and distribution processes, to name a few [1, 7, 8, 11, 30, 37, 48]. Efficient scheduling of these activities can significantly contribute to cost reduction, optimize the use of limited resources, reduce waiting times, and improve service quality.

Among the various scheduling problems, the Parallel Machine Scheduling Problem (PMSP) involves scheduling a set of jobs on a set of machines, considering that each job has an execution time and a setup time between jobs [1, 8, 48]. Multiple objective functions can be used to optimize the problem, such as minimizing job execution time, minimizing job tardiness, and minimizing setup times. This problem, classified as NP-hard, has been extensively studied in scientific literature due to its combinatorial complexity and practical relevance [1, 8, 37]. However, traditional deterministic models assume perfect knowledge of processing and setup times, an assumption that is rarely met in real-life applications. In practice, execution times are subject to variations caused by multiple factors, including equipment breakdowns, changes in resource availability, and variability in human performance, among others. This uncertainty can significantly compromise the feasibility and quality of solutions obtained through deterministic approaches.

This research focuses on the Chance-Constrained Parallel Machine Scheduling Problem (CC-PMSP), a version of PMSP that incorporates uncertainty in job execution and setup times. The problem seeks to assign jobs to different machines, guaranteeing with a certain probability that all assigned jobs can be completed within the available machine time. Given the computational challenges of solving the CC-PMSP, we propose a decomposition approach that divides the problem into two levels. The master problem handles job assignment to machines, determining which jobs will be processed by each machine. The subproblems, one for each machine and scenario, determine the optimal processing sequence of the assigned jobs and verify the feasibility of the solution under the chance constraint.

One of the main drawbacks of such decomposition is that the subproblems correspond to NP-hard problems (i.e., sequence problems with setup times). This work proposes Decision Diagrams (DDs), which are graphical structures employed in optimization to represent the feasibility set of discrete optimization problems [6], to solve these subproblems, aiming to leverage their high reusability across different subproblems. These DDs have been utilized in numerous applications, particularly in scheduling and sequencing problems, with high success [18]. Our decomposition encodes the subproblem using a DD and generates cuts from it to guide the master problem toward optimal solutions. We propose two different DD formulations to address this challenge. The first one, called DD-LastJob (DD-LJ), employs a linear cost

function to evaluate paths in the diagram. The second, called DD-JobSet (DD-JS), employs a non-linear cost function that significantly reduces the diagram's size. Additionally, we develop mechanisms to generate No-Good and Irreducible Infeasible Set (IIS) cuts based on our DDs, which efficiently eliminate infeasible solutions from the search space.

In summary, the main contributions of this work are the following:

1. We present a novel variant of the CC-PMSP and propose a decomposition methodology based on DDs.
2. We propose two DD formulations (DD-LJ and DD-JS) to efficiently solve the subproblems and present novel algorithms to generate No-Good and IIS cuts based on such DDs.
3. We adapt the methodology proposed by Lozano & Smith [42] for two-stage stochastic programming problems to the CC-PMSP, which can be easily extended to other chance-constrained problems.
4. We provide extensive numerical experiments and empirical evidence demonstrating that the proposed DD-based methodology outperforms Integer Programming (IP) alternatives.

The remainder of the paper is organized as follows. Section 2 establishes our research foundation with a literature review, followed by the description and formalization of the CC-PMSP in Section 3. Section 4 introduces our decomposition strategy, while Section 5 reviews DDs and presents the DD-LJ and DD-JS formulations for CC-PMSP subproblems. Section 6 details algorithms for generating No-Good and IIS cuts based on these DDs, followed by our computational experiments and results analysis in Section 7. Finally, Section 8 presents our findings, draws conclusions, and outlines directions for future research.

## 2 Literature Review

This research studies chance-constrained programming (CCP) problems with binary variables. Specifically, we consider problems with the following structure

$$\max \quad \mathbf{u}^\top \mathbf{x} \tag{1a}$$
$$\text{s.t.} \quad \mathbf{A}\mathbf{x} \leq \mathbf{b} \tag{1b}$$
$$\mathbb{P}(\exists \mathbf{y} \in \mathscr{Y}(\mathbf{x}, \omega)) \geq 1 - \varepsilon \tag{1c}$$
$$\mathbf{x} \in \{0, 1\}^n, \tag{1d}$$

where the objective function (1a) seeks to maximize the sum of utilities associated with the solution, subject to deterministic linear constraints (1b), a chance constraint (1c) ensuring feasibility with probability at least $1 - \varepsilon$ ($\varepsilon \in (0, 1)$) with respect to realizations $\omega \in \Omega$ of random parameters, and binary constraints (1d). We now review the literature related to CCP problems and DDs.

## 2.1 Chance-Constrained Programming Problems

CCP problems have been studied since at least the work of [49], but remain computationally challenging for two main reasons: (i) the solution space is not convex, and (ii) evaluating the feasibility of a solution requires integrating multi-dimensional functions [43]. Traditional methods only achieve good solutions in special cases. For example, if the chance constraint is a linear inequality where the random coefficients follow a normal distribution, it is possible to reformulate the problem as a deterministic non-linear convex problem if $\varepsilon < 0.5$ [9]. Another example is for problems with $\mathscr{Y}(\mathbf{x}, \omega) = \{\mathbf{y} : T\mathbf{x} + W\mathbf{y} \geq h(\omega)\}$, where $h(\omega)$ follows a discrete finite distribution, which can be solved with a IP formulation [33].

Luedtke [43] proposed a formulation to exactly solve problems with chance constraints, considering discrete distributions with finite support and random polyhedral constraints. Their procedure decomposes the problem into a master problem containing all the deterministic constraints and one subproblem for each scenario, and then solves the problem using a branch-and-cut algorithm. Although general, this procedure is typically tailored to each application, depending on the structure of the subproblem and the nature of its variables. Liu et al. [38] extended this formulation to include second-stage decisions with additional costs and consideration of the magnitude of not meeting certain scenarios. Building on these foundations, Elçi & Noyan [21] further extended these formulations to optimize humanitarian aid distribution and capacity planning for future disasters by incorporating risk measures through CVaR. Similarly, Lodi et al. [39] adapted these approaches by relaxing problem assumptions to enable application in hydroelectric generation planning.

To the best of our knowledge, our work is the first to consider chance constraints in the PSMP with uncertainty in the processing and setup times. Song & Leus [53] defined a variant of the CC-PMSP with uncertain processing times following a normal distribution, and does not consider setup times. Thus, their chance constraints can be reformulated as a second-order conic constraint, which can be handled by commercial solvers. The authors proposed a branch-and-cut approach and dichotomic search procedures to solve the problem more efficiently. In contrast, our CC-PMSP variant considers uncertain processing and setup times drawn from any random distribution with finite support, which makes the problem considerably more challenging, as it incorporates a joint-chance constraint for each machine to address the sequential aspect of the problem.

Researchers have also applied chance constraints to handle uncertainty in other related scheduling problems. Deng & Sheng [20] address a multi-server appointment scheduling problem with chance constraints where appointment times are uncertain with finite support. Similarly to our work, they proposed a two-stage decomposition to efficiently solve the problem, where their decomposition strongly relies on the packing structure of their problem. Additionally, similar to [53], the authors do not consider uncertain setup times, which simplifies their subproblems compared to ours. Additional related works include applications of chance constraints to scheduling problems, such as project scheduling [55], surgery planning [47], workforce scheduling [22], job-shop scheduling [52], and multi-depto vehcile scheduling [15], to name a few. While these applications are somewhat related to our problem, we note that

none of these works consider a joint chance constraint similar to the one proposed for our CC-PMSP variant.

## 2.2 Decision Diagrams

Decision Diagrams (DD) are graph-based structures with many applications. Initially introduced for circuit design [29, 36], DDs have also proven helpful for sequential pattern mining and genetic programming [40, 56]. In optimization, DDs have been shown to be useful and versatile tools for solving various discrete problems with mathematical programming techniques [5, 13]. For instances, DDs can be employed in cut generation algorithms [2, 3, 12, 19, 54], in global constraints [28], in bound computations [4, 10, 14, 26, 45, 51] and in handling stochastic problems [35, 42].

Several studies utilize DDs to solve scheduling problems, beginning with Cire & van Hoeve [18] who proposed a general approach for the sequencing problem. Since this seminal work, several researchers have employed DDs to solve scheduling and sequence problems (e.g., vehicle routing variants [31, 16], and clinical rotation optimization [17], to name a few). Indeed, several works have tackled deterministic variants of the PMSP. Lozano et al. [41] to solve PMSP, where jobs consist of two tasks with a minimum time interval between them. Also, Kowalczyk & Leus [32] implemented branch-and-price based on DDs for scheduling problems with parallel machines, and Raghunathan et al. [50] used a similar algorithm to optimize time and number of transfers for first and last mile of a transportation system. We refer the reader to the survey [13] and tutorial [27] for more DD applications.

While DDs have been extensively used in deterministic problems, there is limited work on solving problems with uncertain parameters. Lozano & Smith [42] proposed a solution approach to a special type of two-stage stochastic integer problems using DDs with their novel incorporation of capacitated arcs. This idea enables parameterizing arc availability as a function dependent on a binary vector **x** and generates Benders' cuts in a standard two-stage decomposition approach. MacNeil & Bodur [44] extended this DD-based approach, allowing for a greater variety of two-stage stochastic problems where the first stage influences second-stage constraints. They argue that using capacitated arcs can lead to poor DD reductions and propose a new parameterization based on arc costs. Guo et al. [24] adopted the cuts presented by [42] to propose a decomposition to solve the Stochastic Distributed Operating Room Scheduling problem.

To the best of our knowledge, there are only two other works that have utilized DD for CCP problems. Latour et al. [34, 35] employ DDs to represent decisions and the probabilistic distributions of the problem. Their DD is reformulated into a quadratic constraint model, which is then linearized and introduced into an IP formulation of the original problem. Haus et al. [25] also use DDs to solve two-stage stochastic problems where first-stage decisions affect the stochastic process (i.e., endogenous uncertainty), which are then used to reformulate the chance constraints into linear inequalities. To our knowledge, no other techniques have been developed to solve scheduling problems with chance constraints similar to the CC-PMSP proposed in this work (i.e., a decomposition approach where subproblems are solved using DDs).

## 3 Problem Description and Formulation

The CC-PMSP involves deciding the assignment and order of a set $\mathscr{J}$ of jobs on a set $\mathscr{M}$ of machines. Each job $j \in \mathscr{J}$ has an assigned utility $f_j$ that dictates its importance in scheduling, as well as execution times $t_j$ and setup times $d_{jk}$ with respect to another job $k \in \mathscr{J}$. These times, both execution and setup, are stochastic and are not known at the time of optimization. We will identify this stochasticity with a possible realization or scenario $\omega \in \Omega$ for the rest of the document, and therefore we write $t_j^{\omega}$ and $d_{jk}^{\omega}$ as realizations of these random parameters.

The machines in set $\mathscr{M}$ are homogeneous, meaning they are all identical, and thus do not influence the different execution times $t_j^{\omega}$ and setup times $d_{jk}^{\omega}$ of each job $j, k \in \mathscr{J}$ for any realization $\omega \in \Omega$. Each machine has a time availability of $T$ and a maximum capacity of $B$ assigned jobs. The execution time and setup time of jobs assigned to a machine must meet the availability of time with a probability of at least $1 - \varepsilon$, with $\varepsilon \in (0, 1)$.

It is important to mention that this type of problem is not limited exclusively to solving CC-PMSP. The problem formulation is general enough to be used in other contexts with the same parameters and sets. For example, it can solve the Chance-Constrained Operating Room Scheduling (CC-ORS) problem where various surgical operations are assigned and sequenced in different operating rooms. Each of these operations has an execution time, which is not known with certainty because it may become complicated or may turn out easier than expected, and a stochastic setup time between operations, corresponding to the cleaning of the operating room and the preparations needed for the next intervention. The operating rooms also have a limited time and a maximum capacity of operations that can be assigned.

Another example that can be solved is the Chance-Constrained Vehicle Routing Problem (CC-VRP), where, instead of assigning and sequencing jobs to machines, different home deliveries are assigned and sequenced to different vehicles. Each of these deliveries has a stochastic execution time, which corresponds to the time it takes for the customer to receive the package, and a setup time that represents the travel time between customers. Each vehicle also has a limited time of use (for example, given the driver's work day) and a maximum capacity of orders to deliver.

What is interesting in these examples is the difference in the distribution of execution and setup times. For the CC-ORS problem, the execution time of a surgical operation is considerably longer than the setup time for the next operation. On the contrary, for the CC-VRP, the travel time between orders is significantly longer than the time it takes for the customer to receive the package. These two problems together allow us to perform more robust experiments, covering various configurations to evaluate the new formulations proposed in this research.

### 3.1 Formulation with Chance Constraints

One way to model the CC-PMSP is with the decision vector $\mathbf{x} \in \{0, 1\}^{|\mathscr{J}| \times |\mathscr{M}|}$, where each variable $x_{jm}$ indicates whether job $j \in \mathscr{J}$ is assigned to machine $m \in \mathscr{M}$. The formulation is as follows:

$$\max \quad \sum_{j \in \mathscr{J}} \sum_{m \in \mathscr{M}} f_j x_{jm} \tag{2a}$$

$$\text{s.t.} \quad \sum_{m \in \mathscr{M}} x_{jm} \leq 1 \qquad \forall j \in \mathscr{J} \tag{2b}$$

$$\sum_{j \in \mathscr{J}} x_{jm} \leq B \qquad \forall m \in \mathscr{M} \tag{2c}$$

$$\mathbb{P}(\exists \mathbf{y} \in \mathscr{Y}(\mathbf{x}, \omega)) \geq 1 - \varepsilon \tag{2d}$$

$$x \in \{0,1\}^{|\mathscr{J}| \times |\mathscr{M}|} \tag{2e}$$

The objective function (2a) maximizes the utilities of each job assigned to a machine. Constraints (2b) ensure that no job is assigned to multiple machines. Constraints (2c) impose that each machine is not assigned more jobs than its capacity limit.

Constraint (2d) guarantees that $\mathscr{Y}(\mathbf{x}, \omega)$ is nonempty with probability at least $1 - \varepsilon$. In our problem, $\mathscr{Y}(\mathbf{x}, \omega)$ encodes sequencing the jobs assigned to each machine by $\mathbf{x}$ so that, given scenario $\omega$ of execution and setup times, the available time $T$ is not exceeded. The set $\mathscr{Y}(\mathbf{x}, \omega)$ is composed by vectors of variables $y_{jk}^{m\omega} \in \{0,1\}$ that indicate whether job $j \in \mathscr{J} \cup \{0\}$ precedes job $k \in \mathscr{J} \cup \{0\}$ on machine $m \in \mathscr{M}$ and a vector $\mathbf{u}^{m\omega}$ that ensures that the sequences of jobs assigned to all machines are valid. The set is defined as follows:

$$\mathscr{Y}(\mathbf{x}, \omega) = \quad \{\mathbf{y} \in \mathbb{R}^{(|\mathscr{J}|+1) \times (|\mathscr{J}|+1) \times |\mathscr{M}|} :$$

$$\sum_{k \in \mathscr{J} \cup \{0\}} y_{jk}^{m\omega} = x_{jm} \qquad \forall j \in \mathscr{J}, m \in \mathscr{M} \tag{3a}$$

$$\sum_{j \in \mathscr{J} \cup \{0\}} y_{jk}^{m\omega} = x_{km} \qquad \forall k \in \mathscr{J}, m \in \mathscr{M} \tag{3b}$$

$$\sum_{k \in \mathscr{J} \cup \{0\}} y_{0k}^{m\omega} = 1 \qquad \forall m \in \mathscr{M} \tag{3c}$$

$$\sum_{j \in \mathscr{J} \cup \{0\}} y_{j0}^{m\omega} = 1 \qquad \forall m \in \mathscr{M} \tag{3d}$$

$$y_{jj}^{m\omega} = 0 \qquad \forall j \in \mathscr{J}, m \in \mathscr{M} \tag{3e}$$

$$\exists \mathbf{u}^{m\omega} \in \mathscr{S}_m(\mathbf{y}, \omega) \qquad \forall m \in \mathscr{M} \tag{3f}$$

$$y_{jk}^{m\omega} \in \{0,1\} \qquad \forall j, k \in \mathscr{J} \cup \{0\}, m \in \mathscr{M}\} \tag{3g}$$

The set employs the index 0 as the dummy node used in scheduling problems to refer to the start and end of the sequence to be assigned to the jobs. Constraints (3a) and (3b) ensure that every job $j \in \mathscr{J}$ that has been assigned to machine $m \in \mathscr{M}$, is also assigned a position in the sequence of this machine. If $j$ has not been assigned to $m$, then the constraints ensure that this job is not in the machine's sequence. Constraints (3c) and (3d) ensure that the dummy node 0 represents both the start and end of the job sequence, ensuring that there exists one and only one job associated with this start and end. Constraints (3e) limit that a job $j \in \mathscr{J}$ assigned to machine $m \in \mathscr{M}$ does not precede the same job $j$.

Constraints (3f) guarantee a unique order $o$ in the sequence of jobs. We utilize the Single Commodity Flow (SCF) formulation [23] to model the scheduling problem. In our context, this formulation includes the following equations:

$$\mathscr{S}_m^{SCF}(\mathbf{y}, \omega) = \{\mathbf{u}^{m\omega} \in \mathbb{R}^{(|\mathscr{J}|+1) \times (|\mathscr{J}|+1)} :$$

$$\sum_{k \in \mathscr{J} \cup \{0\}} u_{jk}^{m\omega} - \sum_{k \in \mathscr{J} \cup \{0\}} u_{kj}^{m\omega} = \sum_{k \in \mathscr{J} \cup \{0\}} y_{jk}^{m\omega}(t_j^{\omega} + d_{jk}^{\omega}) \qquad \forall j \in \mathscr{J} \quad (4a)$$

$$u_{jk}^{m\omega} \leq T y_{jk}^{m\omega} \qquad \forall j,k \in \mathscr{J} \cup \{0\} \quad (4b)$$

$$u_{jk}^{m\omega} \geq 0 \qquad \forall j,k \in \mathscr{J} \cup \{0\}\} \quad (4c)$$

The vector $\mathbf{u}^{m\omega} \in \mathbb{R}^{(|\mathscr{J}|+1) \times (|\mathscr{J}|+1)}$ acts as a time counter for the sequence of jobs assigned to each machine, so each variable $u_{jk}^{m\omega}$ indicates the time at which job $k \in \mathscr{J} \cup \{0\}$ can begin on machine $m \in \mathscr{M}$, taking into account the execution time of job $j \in \mathscr{J} \cup \{0\}$ and the setup time needed between them in scenario $\omega \in \Omega$.

Constraints (4b) control that any pair of jobs $(j,k)$, assigned in that order to machine $m$, cannot exceed the time limit $T$ of said machine. Constraints (4a) define the values of each variable within vector $\mathbf{u}^{m\omega}$, taking into account the accumulated time prior to job $j$, its execution time, and the setup time needed for the next job $k$ on machine $m$. Finally, (4c) imposes that the variables are non-negative, and together with (4b), force that the time of any pair of jobs $(j,k)$ that is not assigned to machine $m$ is equal to 0.

We also explored the Miller-Tucker-Zemlin (MTZ) formulation [46] as an alternative approach, but ultimately chose SCF due to its superior performance in preliminary experiments. The complete formulation and constraints of the MTZ formulation can be found in Appendix A.

## 3.2 Formulation by Scenarios

From now on, we assume that $\Omega$ is finite. Given that there are multiple scenarios $\omega \in \Omega$, we sum the probabilities associated with each scenario in which the scheduling problem constraints are satisfied and force this sum to be greater than or equal to the desired probability, $1 - \varepsilon$. For this, we introduce a new vector of variables $\mathbf{z} \in \{0,1\}^{|\Omega|}$ in which each $z^{\omega} \in \{0,1\}$ indicates whether the scheduling constraints are satisfied in scenario $\omega$. Thus, the chance constraint takes the following form:

$$\sum_{\omega \in \Omega} p^{\omega} z^{\omega} \geq 1 - \varepsilon$$

where $p^{\omega}$ represents the probability of occurrence of scenario $\omega \in \Omega$. We consider that all scenarios are equally probable, so each $p^{\omega}$ takes the value of $\frac{1}{|\Omega|}$.

The complete model with all deterministic constraints can be found in Appendix B. Although a deterministic equivalent model provides an adequate representation of the problem, its complexity increases exponentially with the number of jobs, machines, and scenarios. Without proper decomposition, it becomes not possible to computationally solve this model. Therefore, in the following section, we present

the approach we employed to avoid the limitations of the deterministic equivalent model.

## 4 Decomposition Approach

This section presents a decomposition for the CC-PMSP, in which the master problem assigns jobs to different machines, and the subproblems verify whether a job sequence exists that satisfies the time limit for each scenario and machine. Specifically, the master problem (MASTER) is defined as follows:

$$\max \sum_{j \in \mathcal{J}} \sum_{m \in \mathcal{M}} f_j x_{jm} \qquad \text{(MASTER)}$$

$$\text{s.t.} \sum_{m \in \mathcal{M}} x_{jm} \leq 1 \qquad \forall j \in \mathcal{J} \qquad (6a)$$

$$\sum_{j \in \mathcal{J}} x_{jm} \leq B \qquad \forall m \in \mathcal{M} \qquad (6b)$$

$$\sum_{\omega \in \Omega} p^\omega z^\omega \geq 1 - \varepsilon \qquad (6c)$$

$$x_{jm} \in \{0,1\} \qquad \forall j \in \mathcal{J}, m \in \mathcal{M} \qquad (6d)$$

$$z^\omega \in \{0,1\} \qquad \omega \in \Omega \qquad (6e)$$

As detailed in Section 3, variables $\mathbf{x}$ assign jobs to machines and $\mathbf{z}$ identify if a scenario satisfies the scheduling constraints. The objective function seeks to maximize the utilities of the assigned jobs. Constraints (6a) ensure that a job is not assigned to more than one machine, and constraints (6b) enforce that the machine limits are met. Lastly, (6c) encodes the chance constraint as discussed in Section 3.2.

As mentioned earlier, the subproblems are responsible for planning and ordering the jobs assigned to each machine. Note that the independence between machines allows for the decomposition to be not only in the set $\Omega$, but also in the set $\mathcal{M}$, so each subproblem $\mathcal{Q}(\hat{x}, \omega, m)$ depends on the candidate solution $\hat{x}$, the scenario $\omega \in \Omega$, and the machine $m \in \mathcal{M}$ to which the set of jobs was assigned. There are several possible ways to solve these subproblems, including the two DD-based formulations presented in Section 5. A possible IP formulation (SubIP) for the subproblems is:

$$\mathcal{Q}(\hat{x}, \omega, m) = \max z \qquad \text{(SubIP)}$$

$$\text{s.t.} \sum_{k \in \mathcal{J} \cup \{0\}} y_{jk}^{m\omega} = \hat{x}_{jm} \qquad \forall j \in \mathcal{J} \quad (7a)$$

$$\sum_{j \in \mathcal{J} \cup \{0\}} y_{jk}^{m\omega} = \hat{x}_{km} \qquad \forall k \in \mathcal{J} \quad (7b)$$

$$\sum_{k \in \mathcal{J}} y_{0k}^{m\omega} = 1 \qquad (7c)$$

$$\sum_{j \in \mathcal{J}} y_{j0}^{m\omega} = 1 \qquad (7d)$$

$$y_{jj}^{m\omega} = 0 \qquad \forall j \in \mathcal{J} \quad (7e)$$

$$u_{jk}^{m\omega} - y_{jk}^{m\omega}(T - t_j^{\omega}) \leq M(1-z) \qquad\qquad \forall j,k \in \mathscr{J} \cup \{0\} \quad (7f)$$

$$\sum_{k \in \mathscr{J} \cup \{0\}} u_{jk}^{m\omega} - \sum_{k \in \mathscr{J} \cup \{0\}} u_{kj}^{m\omega} = \sum_{k \in \mathscr{J} \cup \{0\}} y_{jk}^{m\omega}(t_j^{\omega} + d_{jk}^{\omega}) \qquad\qquad \forall j \in \mathscr{J} \quad (7g)$$

$$\mathbf{y}^{m\omega} \in \{0,1\}^{(|\mathscr{J}|+1)\times(|\mathscr{J}|+1)} \qquad\qquad\qquad (7h)$$

$$\mathbf{u}^{m\omega} \geq 0 \qquad\qquad\qquad\qquad (7i)$$

$$z \in \{0,1\} \qquad\qquad\qquad\qquad (7j)$$

The constraints are the same as in $\mathscr{Y}(\mathbf{x}, \omega)$ and $\mathscr{S}_m^{SCF}(\mathbf{y}, \omega)$.

Given that we want to know if the subproblem is feasible, the objective function of $\mathscr{Q}(\hat{x}, \omega, m)$ indicates if it is feasible (i.e., $z = 1$) when there exists a job sequence that satisfies the limit $T$. If it is feasible, then the set of jobs that was assigned to machine $m \in \mathscr{M}$ is a possible valid configuration in scenario $\omega \in \Omega$. If it is not feasible, it means that there is no order for the assigned jobs that meets the machine's availability in scenario $\omega$, so one or more cuts should be added to the master problem.

The cuts we will use for the master problem have the following form:

$$\sum_{j \in \bar{\mathscr{J}}^h} x_{jm} \leq |\bar{\mathscr{J}}^h| - z^{\omega} \qquad \forall h \in \mathscr{H}^{\omega}, \forall m \in \mathscr{M} \qquad (8)$$

where $\mathscr{H}^{\omega}$ is the set of machines that, in scenario $\omega \in \Omega$, gave infeasible results with the candidate solution $\hat{x}$, and $\bar{\mathscr{J}}^h$ is a set of jobs that led to such infeasibility in machine $h \in \mathscr{M}$. Therefore, inequality (8) ensures that the next candidate solution changes at least one job from the sets of jobs that were infeasible in past solutions. We use two types of cuts in our experimental evaluation: No-Good and IIS. In the case of a No-Good cut, the set $\bar{\mathscr{J}}^h$ contains all jobs assigned to $h$, that is, $\bar{\mathscr{J}}^h = \{j \in \mathscr{J} : \hat{x}_{jh} = 1\}$. On the other hand, the set $\bar{\mathscr{J}}^h$ for the IIS cuts covers the smallest combinations of jobs that make machine $h$ infeasible. Section 6 discusses the cuts in greater depth.

### 4.1 Master Problem Valid Inequalities

We now present a series of valid constraints that we add to the master problem (MASTER) to reduce the number of candidate solutions $\hat{x}$ that lead to equivalent and/or infeasible solutions.

#### 4.1.1 Symmetry Breaking Constraints

The CC-PMSP problem exhibits symmetries in its solutions due to the homogeneity of the machines. Thus, assigning the same set of jobs to one machine or another leads to equivalent solutions; however, the model does not recognize these equivalences, which can cause it to explore redundant solutions. In particular, the model can generate up to $|\mathscr{M}|!$ different solutions, where the sets of assigned jobs only exchange their machines. This redundancy introduces inefficiencies in the process of searching for the optimal solution, thereby increasing the resolution time without contributing to improvements in the final result.

To address this issue, we add the following symmetry-breaking constraints to (MASTER):

$$x_{jm} = 0 \qquad \forall j \in \mathscr{J}, m \in \mathscr{M} : m > j \qquad (9a)$$

$$x_{j(m+1)} \leq \sum_{k \in \mathscr{J}:k<j} x_{km} \qquad \forall j \in \mathscr{J}, m \in \mathscr{M} : m < |\mathscr{M}| \qquad (9b)$$

These constraints establish a lexicographic order in the sets of jobs assigned to the machines, identifying and excluding symmetric solutions. In particular, these constraints impose that the sets of jobs with lower indices are assigned to machines with smaller indices. Constraints (9a) guarantee that the index of a job is always less than the index of the machine to which it is assigned. Meanwhile, constraints (9b) maintain the lexicographic order by preventing a job with a lower index from being assigned to the previous machine, since if this happens, then the previous machine is lexicographically greater than the current machine and they should exchange jobs. We refer the reader to Appendix C for more details on these symmetry-breaking constraints, where an example is provided that illustrates how these constraints effectively eliminate redundant solutions while preserving optimality.

### 4.1.2 Improvements to Candidate Solutions

A challenge identified during preliminary experiments was the large number of candidate solutions that do not satisfy the chance constraint. Each of these infeasible solutions forces us to solve the subproblems and add cuts, which implies a significant increase in the running time of our decomposition. To mitigate this effect, we introduce additional constraints to the master problem with more information about the subproblems. The objective of these constraints is to improve the quality of the generated candidate solutions, thereby reducing the occurrence of infeasible solutions and, consequently, enhancing the efficiency of the resolution process.

We implemented three types of constraints to augment the master problem, aiming to reduce the number of infeasible candidate solutions $\hat{\mathbf{x}}$. The first type of constraints corresponds to a scenario relaxation approach, where we include one constraint per scenario and machine that represents a relaxation of the corresponding subproblem. These additional constraints ensure that the candidate solution $\hat{\mathbf{x}}$ satisfies the time limit of each machine for all scenarios $\{\omega \in \Omega : z^{\omega} = 1\}$ with the most optimistic setup times, that is,

$$\sum_{j \in \mathscr{J}} x_{jm}(t_j^{\omega} + \min_{k \in \mathscr{J}}\{d_{jk}^{\omega}\}) \leq T + M(1 - z^{\omega}) \qquad \forall m \in \mathscr{M}, \omega \in \Omega \qquad (10)$$

For each machine $m \in \mathscr{M}$ and for each scenario $\omega \in \Omega$, the constraint aggregates all execution times and the minimum setup times of the jobs assigned to machine $m$, ensuring that this sum does not exceed the machine's time limit. By considering the minimum of the setup times, this sum represents a lower bound on the time required to complete the entire sequence of jobs assigned to machine $m$ in scenario $\omega$. Consequently, this approach eliminates only those combinations that, even under the

most favorable sequencing conditions, would be unable to satisfy the available time constraint.

We also consider two alternative constraint types: (i) include a small set of scenarios in the master problem, and (ii) introduce an optimistic scenario constraint. Unfortunately, preliminary results showed poor performance on these alternatives; thus, we briefly discussed them in Appendix D. We note that, although less effective for our specific problem instances, these alternative approaches may offer valuable insights for related CCP problems.

## 5 DD-based Models for Sequencing Problem

This section introduces DD models utilized to solve the subproblem of our decomposition approach. We first formally define DDs for optimization problems and present the notation used throughout this paper. We then discuss how to use DDs to model scheduling problems and show the two proposed formulations for solving the subproblems of the CC-PMSP.

### 5.1 Definition of DDs in Optimization

To properly define DDs, consider the following optimization problem where $\mathbf{x}$ is a $k$-dimensional vector of integer decision variables:

$$\min \quad f(\mathbf{x}) \tag{P}$$
$$\text{s.t.} \quad \mathbf{x} \in \mathscr{X}$$

Here $f(\cdot)$ corresponds to a objective function and $\mathscr{X}$ represents the set of constraints that $\mathbf{x}$ must satisfy to guarantee valid solutions. A DD for the problem (P) is a directed acyclic graph $\mathscr{D} = (\mathscr{N}, \mathscr{A})$, where its sets of nodes and arcs are represented by $\mathscr{N}$ and $\mathscr{A}$, respectively. The set of nodes $\mathscr{N}$ is partitioned into $k+1$ layers $\mathscr{L} = \{\mathscr{L}_1, ..., \mathscr{L}_{k+1}\}$, where $\bigcup_{i=1}^{k+1} \mathscr{L}_i = \mathscr{N}$ and $\mathscr{L}_i \cap \mathscr{L}_j = \emptyset$ for all $i \neq j$. The first and last layer always contain a single node called the root node ($\hat{r}$) and terminal node ($\hat{t}$), respectively.

The set of arcs is given by $\mathscr{A}$, where each arc $a = (n, n') \in \mathscr{A}$ connects an output node (i.e., parent node) $n \in \mathscr{N}$ with an input node (i.e., child node) $n' \in \mathscr{N}$. To refer to these nodes, we will use the notation $\text{out}(a) = n$ for the output node and $\text{in}(a) = n'$ for the input node. In general, arcs connect nodes belonging to consecutive layers of the diagram, that is, if $\text{out}(a) \in \mathscr{L}_i$, then $\text{in}(a) \in \mathscr{L}_{i+1}$. We will use $\delta^{in}(n)$ and $\delta^{out}(n)$ to represent the sets of incoming and outgoing arcs of a node $n \in \mathscr{N}$, respectively.

The DD of problem (P) relates each of the layers $\mathscr{L}_i$, $i \in \{1, ..., k\}$, with a decision variable $x_i$ and the outgoing arcs of that layer with the possible values of the variable. We will refer to $\text{val}(a)$ and $\text{c}(a)$ as the value assigned to arc $a \in \mathscr{A}$ and its cost for assigning that value, respectively. It is worth mentioning that no two arcs are leaving the same node that contain the same $\text{val}(a)$.

Each of the paths that can be formed from the root node to the terminal node is a solution to the problem (P). As we pass through the different layers through arcs

$a \in \mathscr{A}$, values $\mathsf{val}(a)$ are assigned to the various variables, and the costs $\mathsf{c}(a)$ of the solution are added up. Therefore, to find the optimal solution, we need to find the path with the minimum cost. If the optimization problem were to maximize its objective function, we would need to find the path with the maximum cost.

To construct a DD for (P), we need a Dynamic Programming formulation of the problem. Specifically, we require four key components: (i) a state space, where we denote the state of node $n \in \mathscr{N}$ as $\mathsf{s}(n)$ and its domain of feasible arc values as $\mathsf{dom}(n)$; (ii) a transition function $\mathsf{tran}(\mathsf{s}(n), \mathsf{val}(a))$ that maps a state $\mathsf{s}(n)$ and arc value $\mathsf{val}(a)$ to a resulting state $\mathsf{s}(n')$; (iii) a cost function that determines the cost $\mathsf{c}(a)$ of an arc based on a state and arc value; (iv) and an initial state $\mathsf{s}(\hat{r})$ from which the DD construction begins.

The DD construction is performed directly through the top-down algorithm construction approach developed by Bergman et al. [4]. The specific algorithm adaptation for our problem is provided in Appendix E. We refer readers to the works of [5, 18, 27] for a more comprehensive analysis of DD construction techniques.

## 5.2 DD for Scheduling Problems

We use two different DD-based models to solve the scheduling problem in our decomposition. In each iteration, each diagram must identify if there exists any order of the assigned jobs that meets the time availability of the machine and, if not, generate a cut for the master problem.

To model the order of jobs assigned to a machine, let us consider any machine $m \in \mathscr{M}$ and the set of jobs assigned to it in the master problem, denoted by $\mathscr{J}^m$. This set has a fixed number of jobs, and, thus, the DD employed to sequence the jobs and verify the feasibility of the subproblem has $|\mathscr{J}^m| + 1$ layers. We use the decision vector $\mathbf{w} \in \mathbb{Z}_+^{|\mathscr{J}^m|}$ to assign each job to a position in the sequence, where the value of the variable $w_p$ indicates the index of the job assigned to position $p \in \{1, .., |\mathscr{J}^m|\}$.

Next, we present two DD-based formulations for solving the subproblems. The first formulation uses a DD with a linear cost function, in which the state of each node is composed of two components, one of which indicates the last job assigned. Due to this characteristic, we call this model DD-LJ, for the acronym DD-LastJob. The second formulation employs a DD that has a more complex cost function, designed to reduce the size of the diagram and simplify the state of the nodes. In this case, the state only contains the set of jobs already assigned. This model is called DD-JS, for the acronym DD-JobSet.

### 5.2.1 Last Job DD-based Model

In DD-LJ, nodes use a state composed of two parts. The first part ($\mathsf{s}_{set}$) consists of the jobs that have already been assigned to some position, while the second part ($\mathsf{s}_{last}$) corresponds to the last job that was assigned a position. Formally, we have $\mathsf{s}(n) := (\mathsf{s}_{set}(n), \mathsf{s}_{last}(n))$ with:

$$\mathsf{s}_{set}(n) := \{w_q | q = 1, ..., (p-1)\} \tag{12a}$$

$$\mathsf{s}_{last}(n) := w_{(p-1)} \tag{12b}$$

where $n \in \mathscr{L}_p$ is a node in $p$-th layer and $p \in \{1,..,|\mathscr{J}^m|\}$ is the position in the job order being assigned. The initial state is given by $\mathsf{s}(\hat{r}) = (\{\}, -1)$, where the -1 is an indicator that there is no last job assigned.

Given a node $n \in \mathscr{L}_p$ and an arc $a = (n, n')$, the state of node $n' \in \mathscr{L}_{p+1}$ consists of the same set of jobs that were already assigned in node $n$, along with the job represented by the arc. Then, the transition function for DD-LJ is:

$$\mathsf{tran}(\mathsf{s}(n), j) := (\mathsf{s}_{set}(n') = \mathsf{s}_{set}(n) \cup \{j\}, \mathsf{s}_{last}(n') = j) \tag{13}$$

where $j \in \mathscr{J}^m$ is a job such that $\mathsf{val}(a) = j$.

For each node $n \in \mathscr{L}_p$, the domain of variable $w_p$ is given by $\mathsf{dom}(n) = \mathscr{J}^m \setminus \mathsf{s}_{set}(n)$. This means that the values that $w_p$ can take from node $n$ correspond to the jobs that are not present in $\mathsf{s}_{set}(n)$.

The cost function for an arc $a \in \mathscr{A}$ and a scenario $\omega \in \Omega$ of the DD-LJ model is:

$$\mathsf{c}(a, \omega) = \begin{cases} t^{\omega}_{\mathsf{val}(a)} & \text{if } \mathsf{out}(a) = \hat{r} \\ d^{\omega}_{\mathsf{s}_{last}(\mathsf{out}(a)),\mathsf{val}(a)} + t^{\omega}_{\mathsf{val}(a)} + d^{\omega}_{\mathsf{val}(a),0} & \text{if } \mathsf{in}(a) = \hat{t} \\ d^{\omega}_{\mathsf{s}_{last}(\mathsf{out}(a)),\mathsf{val}(a)} + t^{\omega}_{\mathsf{val}(a)} & \text{otherwise} \end{cases} \tag{14}$$

That is, the cost of each arc $a \in \mathscr{A}$ is calculated as the setup time between the last job assigned in the ouput node, $\mathsf{s}_{last}(\mathsf{out}(a))$, and the job that the arc represents, $\mathsf{val}(a)$, added to the execution time of the latter. The cost is different when they are arcs that enter the terminal node, as a setup time is added to represent leaving the machine ready for the next day. Additionally, the costs for arcs leaving the root node correspond only to the execution time of the first job since the machine is ready to be used. It is worth noting that this cost function is linear, as it only requires the information of the state of the ouput node $\mathsf{out}(a)$ and the job arc $a$ represents to calculate $\mathsf{c}(a)$ of any $a \in \mathscr{A}$. From now on, since the cost associated with an arc is a sum of times, we will call it the time of the arc.

*Example 1 (DD-LJ)* Let us take a machine $m \in \mathscr{M}$ to which the jobs $\mathscr{J}^m = \{1, 2, 3\}$ have been assigned. In Figure 1, we can see that the DD for this example has 3 layers, plus the final layer $\mathscr{L}_4$ that contains the terminal node. Each of the first three layers $\mathscr{L}_i$ is related to the variable $w_i$, defined at the beginning of the section. The numbers of each arc $a \in \mathscr{A}$ correspond to $\mathsf{val}(a)$, which represent the job that was assigned in the current position.

In $\mathscr{L}_1$, we do not have any job assigned, so the domain of $w_1$ is $\mathsf{dom}(\hat{r}) = \mathscr{J}^m$ and an arc is generated for each job. In $\mathscr{L}_2$, we already have the first job that will be processed by the machine assigned, so the domain of $w_2$ has two possible values and depends on the node $n \in \mathscr{L}_2$. If, for example, we arrive at $w_2$ passing through the node with state $(\{2\}, 2)$, it means that the job with index 2 has already been assigned in the first position of the sequence and that for the second position we can choose between the jobs with indices 1 and 3.

In $\mathscr{L}_3$, the last job that remains to be assigned is assigned, reaching the terminal node that already has all jobs assigned. For instance, if for $w_1$ and $w_2$ the jobs with indices 2 and 3 were chosen, respectively, then necessarily $w_3 = 1$. $\qquad\square$
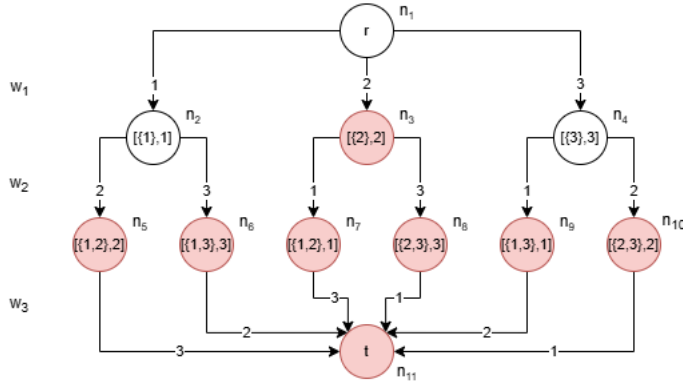
**Fig. 1** DD-LJ structure for a subproblem with three jobs assigned to a machine

### 5.2.2 Job Set DD-based Model

The second DD formulation used in the research was designed to create a smaller DD that represents the same set of solutions. For this, the states of the nodes of this DD are simply the jobs that have already been assigned a position, that is, $s(n) = \{w_q | q = 1, ..., (p-1)\}$, for all $n \in \mathscr{L}_p$. For the construction of this DD, we use an empty set as the initial state (i.e., $s(\hat{r}) = \{\}$). Then, the state of a node $n'$ with incoming arc $a = (n, n')$ corresponds to the set of jobs of the output node $n = \text{out}(a)$ and the job represented in such arc. So the transition function of this DD is

$$\text{tran}(s(n), j) := s(n) \cup \{j\}.$$

Similarly to DD-LJ, the values that $w_p$ can take from node $n \in \mathscr{L}_p$ correspond to the jobs that are not present in the state of the ouput node $s(n)$, i.e., $\text{dom}(n) = \mathscr{J}^m \setminus s(n)$.

Because the state of each node does not contain the index of the last job assigned, the cost function is more complex and dependes on the information provided by arcs in the previous layer [18]. Thus, the cost of each arc is calculated as follows:

$$
c(a, \omega) = \begin{cases}
t^\omega_{\text{val}(a)} & \text{if } \text{out}(a) = \hat{r} \\
\min_{b \in \delta^{in}(n)} \{c(b, \omega) + d^\omega_{\text{val}(b), \text{val}(a)}\} + t^\omega_{\text{val}(a)} + d^\omega_{\text{val}(a), 0} & \text{if } \text{in}(a) = \hat{t} \\
\min_{b \in \delta^{in}(n)} \{c(b, \omega) + d^\omega_{\text{val}(b), \text{val}(a)}\} + t^\omega_{\text{val}(a)} & \text{otherwise}
\end{cases}
\tag{15}
$$

where $a = (n, n')$.

Note that to calculate the cost of each arc $a \in \mathscr{A}$, both the output node, $n = \text{out}(a)$, and the arcs entering it, $\delta^{in}(n)$, must be taken into account. Since the state does not have the last job assigned, it is necessary to recover the value of the setup time $d^\omega_{jk}$ of node $n$. For this, all the jobs that enter node $n$ are reviewed, and the one with the minimum time is selected, taking into consideration the job associated with arc $a$.

This is where linearity is lost, as we must go back to the output node to calculate the cost of an arc. The difference when calculating the cost for arcs that meet $\text{out}(a) = \hat{r}$ or $\text{in}(a) = \hat{t}$ is the same as for the cost calculation in the DD-LJ model. Similarly to DD-LJ, since the cost associated with an arc is the sum of times, we call it the time of the arc.
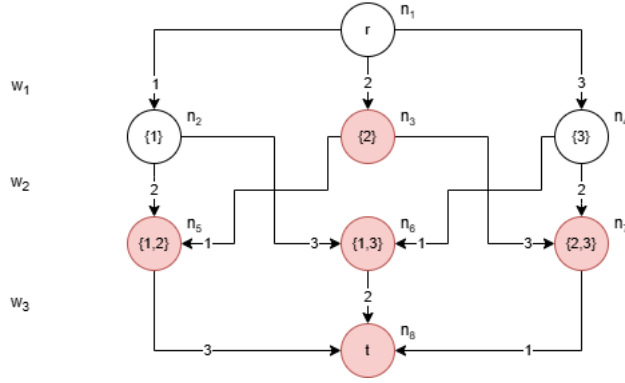


**Fig. 2** DD-JS structure for a subproblem with three jobs assigned to a machine

*Example 2 (DD-JS)* Let us consider the same case as Example 1, that is, a machine with three jobs $\mathscr{J}^m = \{1,2,3\}$. The DD-JS can be seen in Figure 2. We can notice that the first two layers, $\mathscr{L}_1$ and $\mathscr{L}_2$, are identical to those in Figure 1, but layer $\mathscr{L}_3$ changes. Since the state of a node does not depend on the last job assigned, in $\mathscr{L}_3$ we see that there are two ways to reach each node. For instance, if we take the node with state $\{1,2\}$, we can go through the node with state $\{1\}$ and through the node with state $\{2\}$. This not only simplifies the state, but also reduces the size of the DD. As the number of variables increases, this size reduction becomes greater. The red color of the nodes will be discussed in the next section. $\square$

### 5.2.3 Network Flow DD-based Model

Lozano & Smith [42] propose a methodology that uses network flow models over a Binary Decision Diagram (BDD) to address a two-stage stochastic problem, in which the constraints of the second stage can be enabled or disabled depending on the candidate solution generated in the first stage. The decomposition used is a Benders' decomposition, in which the second stage is formulated to be solved with DD.

To adapt the network flow approach for our scheduling problem, we developed two types of DD: Capacitated Binary Decision Diagram (BDD-CAP) and Capacitated Multiple Decision Diagram (MDD-CAP). The former is the adaptation for scheduling problems of the BDD proposed by Lozano & Smith [42], while the latter constitutes our attempt to enhance performance by reducing the size of the DD.

Unfortunately, neither of the network flow-based models demonstrated competitive computational performance for our purposes. Therefore, we have limited our discussion of these approaches in the main body of the paper, and present detailed notations, formulations, and explanations of each model in Appendix F.1. Additionally, Appendix F.2 contains descriptions of the cuts implemented for our problem, adaptations of the strategies proposed by Lozano & Smith [42] to improve cut generation, and Appendix F.3 presents experimental results obtained through these approaches.

## 6 DD-based Cuts

We now detail how to use the DD models introduced in the previous section into our decomposition method and how to generate cuts from them. As previously mentioned, we utilize the DD models (i.e., DD-LJ or DD-JS) to solve each subproblem by verifying whether a job sequence exists with a time smaller than the time limit.

DDs have a great advantage: the structure of the DD only varies as a function of the number of layers that one wishes to construct. In the context of CC-PMSP, since the number of layers $|\mathcal{L}|$ depends on the number of jobs assigned to the machine whose feasibility is to be verified, it is possible to reuse the same DD structure for subproblems where the machine involved has the same number of assigned jobs. This property significantly reduces the time required to solve the subproblems. For a machine $m \in \mathcal{M}$, a DD with $|\mathcal{L}| = |\mathcal{J}^m| + 1$ layers is generated and then, for the remaining subproblems where the number of jobs assigned to a machine coincides with $|\mathcal{J}^m|$, the same DD can be used by only updating the cost function. As a result, instead of building or updating $|\Omega| \cdot |\mathcal{M}|$ subproblems, as happens in the (SubIP) decomposition, we only need to construct at most $B$ diagrams.

Given a candidate solution $\hat{\mathbf{x}}$ from (MASTER), we calculate the minimum sequencing time with one of our DDs for some specific machine and scenario. If this time does not exceed $T$, then the jobs assigned to the machine are feasible and a possible combination for the optimal solution. In case it is greater than $T$, then that job combination does not meet the time constraints for that scenario, and one or more cuts must be added to the master problem.

There are two types of cuts used in this work: No-Good and IIS. For DDs, the calculation of both cuts is relatively simple. In the case of No-Good cuts, the set $\bar{\mathcal{J}}^h$ is the set of jobs assigned to each machine $h \in \mathcal{M}$ that was infeasible in the candidate solution and the cut (8) is created. For example, for the DD in Figure 2, if the arrangement were infeasible, the No-Good cut would simply take the complete set of jobs and generate the following cut for scenario $\omega \in \Omega$:

$$x_{1,m} + x_{2,m} + x_{3,m} \leq 3 - z^{\omega} \qquad \forall m \in \mathcal{M}$$

Additionally, DDs allow calculating several IIS per DD. As its name indicates, an IIS is a set of jobs that correspond to an infeasible assignment and for which any strict subset can lead to a feasible solution. Next, we detail the algorithms for finding IIS in DD-LJ and DD-JS.

## 6.1 IIS Cuts with DD-LastJob Model

We use Algorithm 1 to calculate the IIS sets in the DD-LJ formulation. The shortest times to reach each node are stored in the dictionary `possible_IIS`, where the keys represent sets of jobs, and the values correspond to the minimum scheduling times of those jobs. The algorithm considers that two different nodes $n, n' \in \mathcal{N}$ can share the same set of assigned jobs (i.e., $\mathsf{s}_{set}(n) = \mathsf{s}_{set}(n')$) and that there can be multiple paths to reach the same node. For example, in Figure 1, the nodes with states $\mathsf{s}(n) = (\{1,3\}, 1)$ and $\mathsf{s}(n') = (\{1,3\}, 3)$ share the same set $\mathsf{s}_{set} = \{1,3\}$. To guarantee that $\{1,3\}$ is identified as an infeasible set of jobs, it is necessary to analyze both nodes.

---

**Algorithm 1** IIS Cut Creation for DD-LJ

---

1: **procedure** IISCUT_DD-LJ($\mathscr{D}$, T, $\omega$)
2:     $\mathtt{time}(\hat{r}) = 0$
3:     **for** $l \in \{2, ..., k+1\}$ **do**
4:         **for** $n \in \mathscr{L}_l$ **do**
5:             $\mathtt{time}(n) = \text{Infinity}$
6:             **if** $\mathsf{s}_{set}(n) \notin \mathtt{possible\_IIS}$ **then**
7:                 $\mathtt{possible\_IIS}[\mathsf{s}_{set}(n)] = \text{Infinity}$
8:             **for** $a \in \delta^{in}(n)$ **do**
9:                 $\mathtt{current\_time} = \mathtt{time}(\mathtt{out}(a)) + \mathsf{c}(a, \omega)$
10:                **if** $\mathtt{current\_time} < \mathtt{time}(n)$ **then**
11:                    $\mathtt{time}(n) = \mathtt{current\_time}$
12:                    **if** $\mathtt{time}(n) < \mathtt{possible\_IIS}[\mathsf{s}_{set}(n)]$ **then**
13:                        $\mathtt{possible\_IIS}[\mathsf{s}_{set}(n)] = \mathtt{time}(n)$
14:     $\mathtt{IIS\_SET} = []$
15:     **for** $\mathtt{job\_set}, \mathtt{min\_time} \in \mathtt{possible\_IIS}$ **do**
16:         $\mathtt{isIIS} = \text{True}$
17:         **if** $\mathtt{min\_time} > T$ **then**
18:             **for** $\mathtt{IIS} \in \mathtt{IIS\_SET}$ **do**
19:                 **if** $\mathtt{IIS} \subseteq \mathtt{job\_set}$ **then**
20:                     $\mathtt{isIIS} = \text{False}$
21:                     **break**
22:             **if** $\mathtt{isIIS} = \text{True}$ **then**
23:                 Append $\mathtt{job\_set}$ to $\mathtt{IIS\_SET}$
24:     **return** $\mathtt{IIS\_SET}$

---

Algorithm 1 iterates through all the nodes of the DD layer by layer, as shown in lines 3 and 4, while lines $5-7$ initialize the values of `possible_IIS` and $\mathtt{time}(n)$. Then, in lines 7-13, all the arcs entering the current node $\delta^{in}(n)$ are reviewed, the accumulated times are calculated using vector `time`, and the minimum of them is stored in `possible_IIS`.

Line 14 defines the set that stores the cuts as `IIS_SET`. In lines $15-17$, the algorithm reviews each of the tuples of `possible_IIS`, which contains the job set and their corresponding minimum time, and compares them with the time availability $T$. If a time exceeds this limit, the algorithm evaluates whether the associated set of jobs is not only infeasible, but also irreducible (lines 18-21). If that is the case, then the set of jobs is added to the `IIS` set (lines 22-23). On the contrary, if it is not irreducible, it implies that at least one of its subsets already belongs to `IIS_SET`.

Note that all IISs of a particular size are included in IIS_SET before iterating over the next size.

*Example 3 (IIS Algorithm for DD-LJ)* Let us consider Example 1, the DD in Figure 1, and the scenario $\omega \in \Omega$. Additionally, for this example, let us take the values $T = 5$, $t_1^\omega = 2$, $t_2^\omega = 6$, $t_3^\omega = 3$ and $d_{j,k}^\omega = 1$ for all $j \in \mathscr{J}^m$ and $k \in \mathscr{J}^m \cup \{0\}$.

The process of generating IIS cuts in DD-LJ begins with calculating the minimum time needed to reach each node. In the first layer, as we can see in (14), the times of the arcs $\delta^{out}(\hat{r})$ are $t_{\mathsf{val}(a)}^\omega$, so their values are 2, 6 and 3 for those reaching nodes $n_2$, $n_3$ and $n_4$, respectively. Since the root node has an accumulated time, $\mathtt{time}(\hat{r})$ in the algorithm, of 0, then for this example we have that $\mathtt{time}(n_2) = 2$, $\mathtt{time}(n_3) = 6$ and $\mathtt{time}(n_4) = 3$. The $\mathtt{possible\_IIS}$ dictionary would add, in this layer, the following tuples: $(\{1\}, 2), (\{2\}, 6), (\{3\}, 3)$.

In (14), we can also see that the times of the arcs in the second layer are the sum of the setup time and the execution time. For the arc reaching node $n_5$, whose state is $(\{1,2\}, 2)$, its associated time is $d_{1,2}^\omega + t_2^\omega = 1 + 6 = 7$. Therefore, since $\mathtt{time}(n_2)$ is the time required to reach the ouput node, $n_2$, plus the time of arc $(n_2, n_5)$, in this case, we have that $\mathtt{time}(n_2) + \mathtt{c}((n_2, n_5), \omega) = 2 + 7 = 9$. In the same way, the time required for each of the other nodes in the layer is calculated, which for this example are, 6 for nodes $n_6$ and $n_9$, 9 for node $n_7$ and 10 for nodes $n_8$ and $n_{10}$. Since the $\mathtt{possible\_IIS}$ dictionary stores the minimum time to reach each of the states, in this layer, the following tuples are added: $(\{1,2\}, 9), (\{1,3\}, 6), (\{2,3\}, 10)$.

Then, we have the terminal node in the last layer. Again, with (14) we can calculate the time of each of the arcs reaching $\hat{t}$. The time of arc $(n_5, \hat{t})$ is calculated with $d_{2,3}^\omega + t_3^\omega + d_{3,0}^\omega = 1 + 3 + 1 = 5$ and if we add the time $\mathtt{time}(n_5)$, we have that the accumulated time of $\hat{t}$ when arriving through arc $(n_5, \hat{t})$ is $5 + 9 = 14$. Similarly, when calculating the times of the other arcs, we have, for this case, that they are all 14.

The accumulated time of the terminal node is $\mathtt{time}(\hat{t}) = 14$ and since it is the only node with $\mathtt{s}_{set} = \{1,2,3\}$, the $\mathtt{possible\_IIS}$ dictionary adds the tuple: $(\{1,2,3\}, 14)$.

Finally, the algorithm evaluates the times of each set of jobs stored in $\mathtt{possible\_IIS}$ to determine if any of them exceeds the time limit $T$, which in this case is $T = 5$. In Figure 1, the nodes marked in red represent the infeasible sets. It can be quickly verified that in the first layer, the only set that exceeds the availability of time is $\{2\}$. In $\mathscr{L}_2$, all sets exceed the value of $T$, but not all are IIS. The algorithm, specifically in line 13, identifies that $\{2\}$ is a subset of the sets $\{1,2\}$ and $\{2,3\}$, which implies that these latter ones cannot be considered IIS. On the other hand, the set $\{1,3\}$ has a cost greater than 5, and none of its subsets, $\{1\}$ and $\{3\}$, belongs to the IIS. Consequently, $\{1,3\}$ is added to the $\mathtt{IIS\_SET}$. Finally, the set $\{1,2,3\}$ is analyzed, but since it contains subsets that are already IIS, it is not included in $\mathtt{IIS\_SET}$.

With this, the process of searching for IIS is concluded. As a result, the following cuts are added to the master problem:

$$x_{2,m} \leq 1 - z^\omega \qquad \forall m \in \mathscr{M}$$
$$x_{1,m} + x_{3,m} \leq 2 - z^\omega \qquad \forall m \in \mathscr{M}$$

$\square$

---

**Algorithm 2** IIS Cut Creation for DD-JS

---
 1: **procedure** IISCut_DD-JS($\mathscr{D}$, T, $\omega$)
 2:     IIS_SET = []
 3:     **for** $l \in \{2, ..., k+1\}$ **do**
 4:         **for** $n \in \mathscr{L}_l$ **do**
 5:             superset = False
 6:             **for** IIS $\in$ IIS_SET **do**
 7:                 **if** IIS $\subseteq$ s($n$) **then**
 8:                     superset = True
 9:                     **break**
10:             **if** superset = True **then**
11:                 **continue**
12:             time = Infinity
13:             job_set = s($n$)
14:             **for** a $\in \delta^{in}(n)$ **do**
15:                 **if** c($a, \omega$) < time **then**
16:                     time = c($a, \omega$)
17:             **if** time > T **then**
18:                 Append job_set to IIS_SET
19:     **return** IIS_SET

---

### 6.2 IIS Cuts with DD-JobSet Model

Algorithm 2 is used to calculate the IIS set of DD-JS cuts. The algorithm, in lines 3 and 4, traverses all the layers of the DD in order, evaluating each of its nodes. In lines 5-9, the algorithm verifies if the state of the current node is a superset of an already found IIS. If it is, then the algorithm moves to the next node (lines 10 and 11). If no IIS is a subset of the current node's state, then the algorithm in lines 12-16 calculates the times of all incoming arcs of each node, storing the minimum time to arrive at the set of jobs. Finally, the minimum time is compared with the time limit $T$ (line 17). This minimum value is used because, if it is less than or equal to $T$, there exists at least one sequence that satisfies the time availability constraint. In this case, the set of jobs is not infeasible and, therefore, cannot be part of the IIS_SET. However, if the minimum time exceeds $T$, there is no feasible sequence for the set of jobs, thus fulfilling the definition of infeasibility. Therefore, the set is added to the IIS_SET in line 18 of the algorithm.

As mentioned earlier, for a set of jobs to be considered an IIS, besides being infeasible it must be irreducible. That is, every strict subset of the set must have at least one feasible job sequence. This requirement is guaranteed in lines 5-11 of the algorithm, where it is verified, for each node, if any set of the IIS is a subset of s($n$). In case it exists, the algorithm moves to the next node. In this way, we ensure that any set of jobs in IIS_SET does not have an infeasible subset.

Finally, once all the nodes of the DD have been evaluated, for each set of jobs identified as part of the IIS, the corresponding cuts are generated using equation (8).

*Example 4 (IIS Algorithm for DD-JS)* Let us consider Example 1, the DD in Figure 2, the scenario $\omega \in \Omega$, and let us assume that the times of the nodes painted red exceed $T$. Additionally, let us take the same values as in the previous example: $T = 5$, $t_1^\omega = 2$, $t_2^\omega = 6$, $t_3^\omega = 3$ and $d_{j,k}^\omega = 1$ for all $j \in \mathscr{J}^m$ and $k \in \mathscr{J}^m \cup \{0\}$.

The process of generating IIS cuts in DD-JS starts from the root node, which has time 0. Advancing sequentially, when evaluating the node with state $\{1\}$, whose time ($t_1^{\omega} = 2$) is less than $T = 5$, the progression continues. However, the node with state $\{2\}$ exceeds the value of $T$, since $t_2^{\omega} = 6 > T = 5$. Therefore, its state is incorporated into IIS_SET. Continuing with the node with state $\{3\}$, the time $t_3^{\omega} = 3$ is also below $T$, so the algorithm advances to the next node.

The node with state $\{1,2\}$ contains an element previously included in IIS_SET (specifically $\{2\}$) and consequently, its revision is omitted. The same happens with the node $\{2,3\}$. Then, we have the node with state $\{1,3\}$. Following formula (15), we have that the cost of the arc is $\min\{(t_1^{\omega} + d_{1,3}^{\omega} + t_3^{\omega}), (t_3^{\omega} + d_{3,1}^{\omega} + t_1^{\omega})\} = 6$ which exceeds the value of $T = 5$. Consequently, it is also added to IIS_SET. And since the remaining nodes will always contain some element of IIS_SET, the process of searching for IISs concludes. As a result, the following cuts are added to MASTER:

$$x_{2,m} \leq 1 - z^{\omega} \qquad \forall m \in \mathcal{M}$$
$$x_{1,m} + x_{3,m} \leq 2 - z^{\omega} \qquad \forall m \in \mathcal{M}$$

$\square$

## 6.3 Comparison of IIS Algorithms

Since the nodes of DD-LJ and DD-JS have different state structures, the algorithms for finding IIS differ quite a bit. The main difference is that for DD-LJ there can be several nodes that share the same $s_{set}(n)$. This characteristic increases the complexity of the algorithm, since, to determine the infeasibility of a set of jobs, it is necessary to examine all the nodes that contain it. For this reason, in Algorithm 1, possible_IIS must be created, iterate through all nodes storing the minimum time to reach each set of jobs, and then compare these values with $T$ and verify that the state is irreducible.

Calculating IIS in Algorithm 2 is simpler because DD-JS nodes cannot share the same set of jobs, which corresponds to the complete state of the node, $s(n)$. This property allows us to directly determine whether the minimum time to reach the node is less than or greater than $T$ and, in case it is irreducible, identify it as an IIS. Also, an additional advantage arises in the implementation of Algorithm 2: in certain cases, it is not necessary to traverse the entire DD-JS. If the algorithm finds a node whose state is an IIS, its descendant nodes can be automatically discarded, since they will be infeasible but not irreducible. This technique is not applicable in the implementation of Algorithm 1, as the verification of irreducibility and infeasibility of the set of jobs is performed only after traversing all the nodes.

## 7 Computational Experiments

We now present the experimental design used to evaluate the performance of the different models proposed throughout this research. We describe in detail the generated data sets, the parameters considered in the instances, and the computational environment used for the tests. Subsequently, we analyze the results obtained, examining

both the overall performance of the models and their specific behavior under different configurations. This analysis enables us to understand the comparative advantages of DD-based approaches with traditional integer programming formulations.

### 7.1 Problems and Instance Generation

To rigorously evaluate the proposed models, we designed three data sets with different characteristics, inspired by real-world applications of the CC-PMSP. These sets represent scenarios where the distribution and proportion between execution times and setup times vary significantly, allowing us to examine the robustness of the formulations against different uncertainty structures. Below, we describe each of these sets in detail and the procedure used to ensure relevant mathematical properties such as the triangle inequality in setup times.

*Operation Room Scheduling (ORS).* This data set is designed to address the CC-ORS problem presented in Section 3 and is characterized by operation execution times significantly greater than setup times between operations. To model this behavior, execution times follow a log-normal distribution with a mean of $\mu = 2$ and a standard deviation of $\sigma = 0.6$, while setup times are generated with parameters $\mu = 0.5$ and $\sigma = 0.15$ following the procedure described at the end of this section.

*Vehicle Routing Problem (VRP).* Unlike the CC-ORS case, the CC-VRP problem, also presented in Section 3, has a considerably smaller time to deliver the order to the customer than the travel time of a vehicle between destinations. To reflect this difference, we interchanged the means and deviations used in CC-ORS: execution times follow a log-normal distribution with a mean of $\mu = 0.5$ and a standard deviation of $\sigma = 0.15$, while setup times are generated with parameters $\mu = 2$ and $\sigma = 0.6$.

*Equal.* Finally, we include a case where execution and setup times have the same mean and deviation. For this, we establish a mean of $\mu = 1.25$ and a standard deviation of $\sigma = 0.375$, maintaining the log-normal distribution for execution times and applying the procedure below to generate setup times.

   To ensure that assigning more jobs to a machine does not decrease the total process time, we require that the setup times satisfy the triangle inequality. For this, we generate $|\mathscr{J}| + 1$ points in a plane, maintaining an average distance of $\mu$ between them. Then, to generate uncertainty, in each scenario, a new point is generated from each original point by displacing it according to a random vector whose modulus follows a half-normal distribution with parameter $\sigma$ and whose angle is uniformly distributed in the interval $[0, 2\pi]$. The distances between each of these new points are the setup times for the scenario.

### 7.2 Instance Parameters

Our experiments consider nine configurations with different combinations of total number of jobs and machines, as well as different ratios $\frac{|\mathscr{J}|}{|\mathscr{M}|}$, to cover a wide range

of situations and evaluate the robustness of the proposed models. These configurations allow us to analyze the behavior of each model under various conditions and understand its performance as a function of the relationship between jobs and machines. The chosen combinations are:

$$(|\mathscr{J}|, |\mathscr{M}|) \in \{(60,6),(80,8),(100,10),(72,6),(96,8),$$
$$(120,10),(84,6),(112,8),(140,10)\}$$

To examine whether any model is favored by a larger number of jobs per machine, experiments with differentiated proportions are designed. The first three combinations present a ratio of 10 jobs per machine, the next three increase this ratio to 12 jobs per machine, and the last three reach 14 jobs per machine, thus allowing a more detailed evaluation of the impact of this relationship on model performance.

We consider the distribution of the data to define the value of the parameter $T$ (i.e., the time limit for each machine). In all data types, the sum of the averages of execution times and setup times is 2.5. With this information, we approximated that on average each job will take 2.5, counting its execution and the setup for the next job. Given that $B$ represents the maximum number of jobs that can be assigned to a machine, it was estimated that, on average, each machine would require a time of $T = B * 2.5$. We introduce the parameter DIF to regulate the difficulty of the problem, which allows us to adjust $T$ by either reducing it to increase complexity or increasing it to facilitate resolution. Thus, $T$ was parameterized as $T = 2.5B + \text{DIF} * 0.3$.

If the value of DIF is too high, the time assigned to each machine increases, which trivializes the assignment and ordering of jobs. If, instead, DIF is very negative, the time constraint becomes stricter, significantly increasing the difficulty of the problem and making it difficult to solve within the 20-minute limit. To ensure that the instances are neither too easy nor excessively complex, DIF values are empirically selected to find intermediate solutions between these two extremes. The values that offered better results are: $\{0.2, 0.25, 0.3\}$ for ORS data, $\{-1, -0.95, -0.9\}$ for VRP data, and $\{-0.3, -0.25, -0.2\}$ for Equal data.

For the experiments, we calculate the big-M parameter $M$ by summing the execution time and the highest setup time for each job. Then, we selected the $B$ jobs with the highest values in this partial sum and added their resulting values to obtain the final estimate of $M$, that is:

$$M = \sum_{j \in \mathscr{J}_{topB}} \left( t_j + \max_{k \in \mathscr{J}: j \neq k} \{d_{ij}\} \right)$$

where $\mathscr{J}_{topB}$ are the $B$ jobs with the greatest duration.

For the remaining parameters, we consider $|\Omega| = 100$, $\varepsilon = 0.05$, and $B = \frac{|\mathscr{J}|}{|\mathscr{M}|}$ for all the experiments. These values were empirically determined with the objective of considering the largest possible instances that do not solve trivially and that, at the same time, present some probability of reaching an optimal solution within the 20-minute limit established for all experiments. A more detailed analysis of the choice of these parameters could be explored in future work.

Overall, the experimental design encompasses three types of problems, nine combinations of $(|\mathscr{J}|, |\mathscr{M}|)$ per problem type, and three different values of $T$ for each

one. Therefore, we have 27 distinct configurations for each problem type. We generate five instances for each configuration, resulting in a total of 135 instances per problem type and 405 instances in total.

### 7.3 Computational Setup

The algorithms are programmed in Python 3.11.6 using Gurobi 11.0.3 in the Visual Studio Code editor. All experiments were run on a 13th Gen Intel(R) Core(TM) i5-13600KF CPU with 16 GB of RAM. As mentioned earlier, each approach has a 20-minute time limit to find an optimal solution for an instance, which includes the time required to create all the models (i.e., IP and DDs models). The source code will be made available in a public repository upon publication.

To evaluate the new formulations presented in this work, we implement the DD-based decomposition using the DD-LJ and DD-JS models described in Section 5, using both No-Good and IIS cuts. In addition, we consider an IP-based decomposition where both (MASTER) and (SubIP) are solved using Gurobi's technologies. This IP-based approach also considers a No-Good and an IIS alternative, where the IIS cuts are found using the tools provided by Gurobi. Additionally, preliminary experiments demonstrated that incorporating symmetric-breaking constraints (9) and scenario relaxation constraints (10) was beneficial; therefore, these constraints are also included in the experiments below.

### 7.4 Empirical Results for the DD-based Decomposition

We now review the results of the 405 instances and the six different approaches: four DD-based decompositions (DD-LJ and DD-JS models using No-Good or IIS cuts) and two IP-based decompositions (i.e., using either No-Good or IIS cuts). Figure 3 presents a profile plot comparing the performance of the six approaches as a function of execution time (in seconds) and Gurobi's gap in percentage. On the X-axis, a combined scale is observed, where values to the left of the vertical line represent time in seconds, while values to the right correspond to Gurobi's gap in percentage. The Y-axis indicates the number of instances solved. Each colored line represents a different model with the type of cut used.

The first black dotted vertical line marks the 20-minute time limit (i.e., 1200 seconds); thus, this first portion of the plot shows the number of instances solved to optimality. From here, we observe that the models that deliver the largest number of optimal solutions are the DD-JS models, then the DD-LJ models, and lastly the IP models. In particular, the DD-JS model with IIS cuts outperforms the other approach, solving 185 instances, while the best IP alternative (i.e., IP with No-Good cuts) only solves 129. Starting from the second black dotted line, the plot counts the instances that did not reach an optimal solution, starting from those that are closest to achieving the optimum (i.e., solutions with a small gap). We observe that the lines representing the DD-based decompositions consistently lie above those of the IP alternatives, indicating, in general, that the feasible solutions generated by the DD-based decompositions are closer to the optimum than those from IP models.
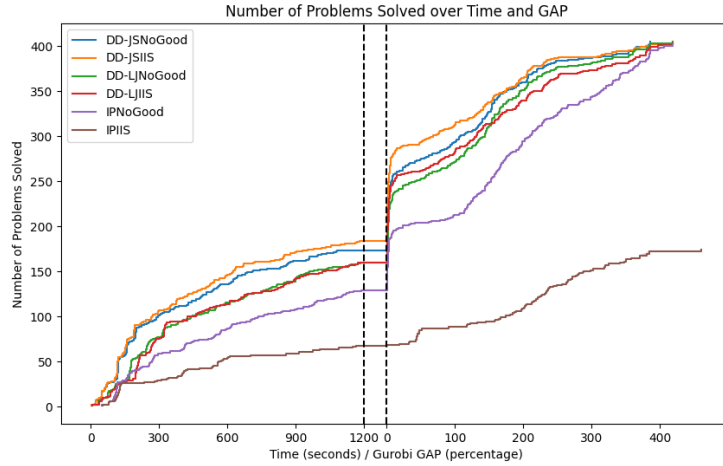
**Fig. 3** Summary Profile Plot of All Experiments

**Table 1** Summary of results for all instances

| Model | Cut | TotalTime | Gap | #Optimal |
|-------|-----|-----------|-----|----------|
| DD-JS | IIS | 814.2 | inf | 184 |
| DD-JS | No-Good | 835.3 | 0.6 | 173 |
| DD-LJ | IIS | 896.5 | inf | 160 |
| DD-LJ | No-Good | 902.5 | 0.7 | 160 |
| IP | IIS | 1113.2 | inf | 64 |
| IP | No-Good | 968.1 | inf | 129 |

Table 1 supports the results presented in Figure 3, where columns "Model" and "Cut" specify the model and the type of cut used, respectively. The "TotalTime" column presents the average execution time, while "Gap" shows the average value of the gap reported by Gurobi. Finally, the "#Optimal" column indicates the number of optimal solutions found within the 20-minute limit. We observe that decompositions based on DD-JS yield a greater number of optimal solutions and require less time compared to the DD-LJ alternative, which, in turn, outperforms the IP models in these metrics. Additionally, it is noteworthy that the DD alternatives with No-Good cuts are the only ones that always obtained a feasible solution, which explains why they are the only models without an infinite gap (i.e., "inf" symbol in the "Gap" column).

Table 2 presents filtered results that eliminate the instances where the models could not find a feasible solution. Here, the "Model" and "Cut" columns have the same meaning as in Table 1. The "OptimalTimes" column represents the average execution time of the solutions that reached the optimum, while "GapWithoutInf" shows us the average gap delivered by Gurobi for instances in which at least one feasible solution is found. Finally, the "#Inf" column indicates the number of instances

**Table 2** Summary of filtered results for all instances (i.e., only feasible solutions)

| Model | Cut | OptimalTimes | GapWithoutInf | #Inf |
|-------|-----|-------------|---------------|------|
| DD-JS | IIS | 350.0 | 0.5 | 1 |
| DD-JS | No-Good | 345.5 | 0.6 | 0 |
| DD-LJ | IIS | 430.7 | 0.7 | 1 |
| DD-LJ | No-Good | 420.1 | 0.7 | 0 |
| IP | IIS | 384.0 | 1.8 | 229 |
| IP | No-Good | 468.3 | 1.2 | 2 |

**Table 3** Detailed results of the decomposition approaches for all instances

| Model | Cut | #Callback | #Cuts | ResolTime | ResolTimePerCB |
|-------|-----|-----------|-------|-----------|----------------|
| DD-JS | IIS | 95.8 | 20550.2 | 676.6 | 7.1 |
| DD-JS | No-Good | 104.8 | 24134.6 | 692.8 | 6.6 |
| DD-LJ | IIS | 76.5 | 16228.6 | 805.8 | 10.5 |
| DD-LJ | No-Good | 92.4 | 21407.3 | 771.8 | 8.4 |
| IP | IIS | 5.2 | 145.8 | 75.1 | 14.4 |
| IP | No-Good | 43.0 | 5237.7 | 653.7 | 15.2 |

| Model | Cut | CreateCutTime | CreateSPTime | UpdateTime |
|-------|-----|---------------|--------------|------------|
| DD-JS | IIS | 0.5 | 0.3 | N/A |
| DD-JS | No-Good | 0.7 | 0.3 | N/A |
| DD-LJ | IIS | 0.4 | 1.7 | N/A |
| DD-LJ | No-Good | 0.6 | 1.6 | N/A |
| IP | IIS | 746.5 | 246.9 | 21.2 |
| IP | No-Good | 0.2 | 179.9 | 115.4 |

where each approach could not find a feasible solution. Taking into account the information in Tables 1 and 2, in the case of DD-JS, we can see that, excluding the single instance where it failed to find a feasible solution, the model with IIS cut is, on average, superior to the No-Good, as it finds more optimal solutions in less time, and the suboptimal solutions have a slightly better gap. For the DD-LJ models, both are very similar, having the same number of optimal solutions and the same gap when excluding the instances that did not find a feasible solution. Finally, the No-Good cuts are far superior to IIS for the IP models.

Table 3 provides insights into why the DD-based approaches perform better compared to IP alternatives. As in the previous tables, the "Model" and "Cut" columns specify the model and the type of cut used, respectively. The "#Callback" and "#Cuts" columns indicate the average number of callbacks that each technique enters and the average cuts created, respectively. The "ResolTime" column represents the sum of the times used in solving each subproblem, while "ResolTimePerCB" shows the division between "ResolTime" and "#Callback". The "CreateCutTime" column indicates the sum of the time used in the creation of cuts, "CreateSPTime" reflects the time dedicated to creating the subproblems, and finally, "UpdateTime" indicates the time used to update the subproblems.

On average, DD techniques enter the callback significantly more times, allowing them to generate a greater number of cuts and, thus, find better solutions. Also, note

**Table 4** Results by Amount of Machine

| Model | Cut | #Machines | TotalTime | Gap | #Optimal |
|-------|-----|-----------|-----------|-----|----------|
| DD-JS | IIS | 6 | 644.1 | inf | 80 |
| DD-JS | IIS | 8 | 812.3 | 0.3 | 65 |
| DD-JS | IIS | 10 | 986.1 | 1.0 | 39 |
| DD-JS | No-Good | 6 | 710.1 | 0.3 | 73 |
| DD-JS | No-Good | 8 | 817.1 | 0.5 | 62 |
| DD-JS | No-Good | 10 | 978.8 | 1.1 | 38 |
| DD-LJ | IIS | 6 | 729.2 | inf | 76 |
| DD-LJ | IIS | 8 | 903.7 | 0.6 | 51 |
| DD-LJ | IIS | 10 | 1056.4 | 1.3 | 33 |
| DD-LJ | No-Good | 6 | 758.8 | 0.4 | 67 |
| DD-LJ | No-Good | 8 | 891.5 | 0.6 | 57 |
| DD-LJ | No-Good | 10 | 1057.2 | 1.2 | 36 |
| IP | IIS | 6 | 961.7 | inf | 36 |
| IP | IIS | 8 | 1170.0 | inf | 15 |
| IP | IIS | 10 | 1208.0 | inf | 13 |
| IP | No-Good | 6 | 733.4 | 0.2 | 72 |
| IP | No-Good | 8 | 1035.0 | 0.9 | 37 |
| IP | No-Good | 10 | 1135.8 | inf | 20 |

that the poor performance of the IP with IIS cuts is primarily due to the time required to generate the cuts, which is four orders of magnitude larger than that of the No-Good alternative. Indeed, this explains why IP with IIS cuts enters fewer times to the callback and generates significantly fewer cuts than all other approaches. In contrast, the DD approaches take fractions of seconds to create the cuts, independently of whether it is a No-Good or an IIS cut. Furthermore, the DD models typically take less time to solve the subproblems than the IP models, with DD-JS being the fastest despite having a non-linear cost function.

Lastly, we observe that as the number of machines increases, the problem becomes more complex. This is evident in Table 4, where an increase in execution time and a decrease in the number of optimal solutions obtained are observed as the number of machines in the instance increases. Likewise, the gap tends to increase, except in cases where a feasible solution is not found.

## 7.5 Empirical Results for Each Problem Type

To provide a more detailed analysis of model performance across different instance types, we present profile plots for three distinct datasets: VRP, ORS, and Equal. These figures allow us to visualize the performance differences between models for each specific dataset. Tables presenting the details of the results, broken down by problem type, are included in Appendix G.

Figure 4 displays the profile plot for the VRP dataset. In this dataset, DD-JS models with both cut types demonstrate superior performance, solving approximately 80 out of 135 instances to optimality before the time limit. The DD-LJ variants follow with 75 and 78 optimal solutions for IIS and No-Good, respectively. In contrast, IP
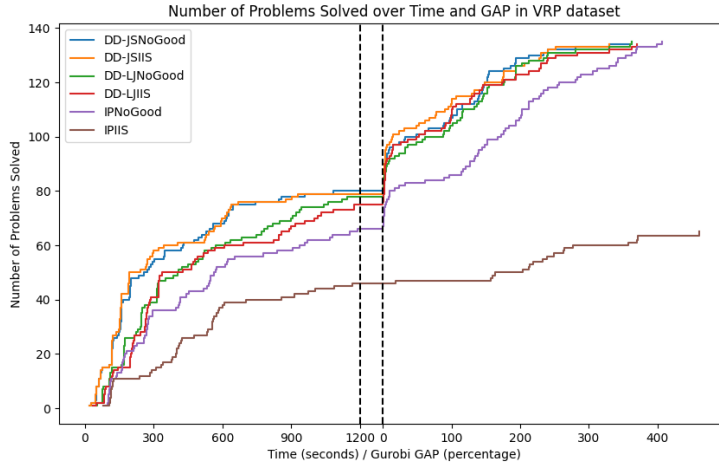
**Fig. 4** Profile Plot for VRP data

models lag significantly, where the approach with No-Good cuts solves 65 instances, and the IIS cut alternative only 45. When considering feasible solutions, the DD-JS models continue to outperform the IP models, with their lines consistently positioned higher in the plot thus giving better sub-optimal solutions. As in Figure 3, IP-IIS shows particularly poor performance in this dataset, with a flatter curve indicating both fewer optimal solutions and larger gaps in non-optimal solutions.

Figure 5 presents the results for the ORS dataset. The performance pattern is similar to that shown in Figures 3 and 4, but the instances are more challenging to solve. DD-JS models achieve optimal solutions for 21 out of 135 instances, while DD-LJ and IP with No-Good cuts solve roughly 18 and 15 instances optimally, respectively. IP with IIS cuts performs particularly poorly in this dataset, finding only one optimal solution. In this plot, we can observe the most dramatic increase after the time limit, where we see that DD models consistently provide better solutions than IP models, with DD-JS notably returning better feasible solutions than DD-LJ.

Figure 6 shows the performance for the Equal dataset, where the most notable feature is the exceptional performance of DD-JS with IIS cuts, which solves 84 out of 135 problems to optimality before the time limit. This is substantially better than other models, with DD-JS No-Good solving 72 instances, DD-LJ with IIS solving 67, DD-LJ with No-Good solving 63, and IP with No-Good solving only 49. IP with IIS shows significantly poorer performance in this dataset, solving fewer than 20 problems optimally. The difference between DD-JS with IIS and other models is particularly striking in this dataset, suggesting that this combination is especially well-suited for problems with equal distributions between realization and setup times. When analyzing the sub-optimal solutions, the DD-JS with IIS model maintains its superiority with a steeper curve, indicating consistently better gap values compared to all other approaches.

**Fig. 5** Profile Plot for ORS data
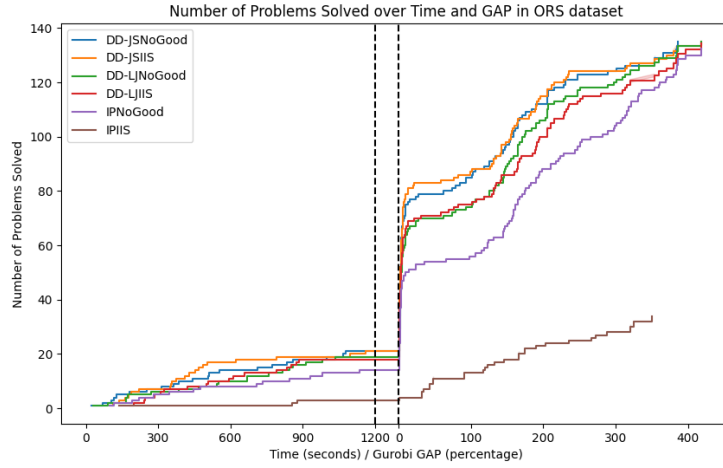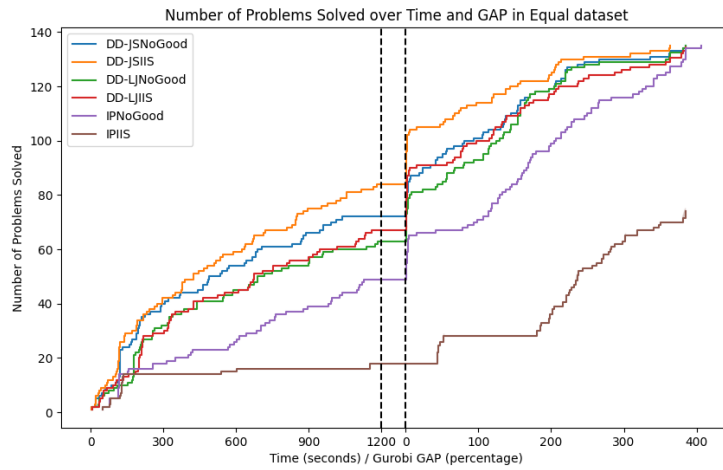


**Fig. 6** Profile Plot for Equal data

## 7.6 Discussion

The consistent performance patterns across these three datasets clearly demonstrate the superiority of DD-based models over IP models, with DD-JS variants generally outperforming DD-LJ variants across all problem types. While the general trend of DD-based superiority holds across all instance types, it is worth noting that the mag-

nitude of this advantage varies, with particularly pronounced differences in the Equal dataset where DD-JS with IIS dramatically outperforms all other approaches.

The results obtained demonstrate that DD-based decompositions significantly outperform IP alternatives in all evaluated datasets, both in the number of optimal solutions found and in the quality of the suboptimal solutions obtained. We identify two main reasons that explain the better performance of the DD-based techniques. First, the construction and update times of DD-based subproblems are considerably faster than those of subproblems formulated using IP. Second, the use of the shortest path algorithm proved to be quicker in solving the subproblems than the IP formulation. This translates to entering the callback more times, which allows for the generation of more cuts and, consequently, higher-quality solutions.

Likewise, the experiments demonstrate that the DD-JS model consistently outperforms the DD-LJ model. This is due to the same reasons mentioned above: the reduction in the size of the DD, in exchange for a more complex cost function, proved to be a beneficial strategy. A more compact DD allows for reducing construction times and shortest path resolution, which translates into being able to enter more callbacks, generate more cuts, and have better solutions.

In general terms, it cannot be categorically stated that IIS cuts outperform No-Good cuts. In the case of the Equal dataset, the approach with IIS cuts can, on average, find a greater number of optimal solutions in less time than models with No-Good cuts. However, in the other datacases, both types of cuts obtained practically identical results in most instances. We consider that this minimal differentiation could be attributed to the nature of the problems we generated, as in these experiments we observe that the IIS cuts are quite similar to No-Good cuts; that is, in most instances both cuts have the same or a similar infeasible set of jobs. This does not favor the IIS cuts because they are more computationally intensive, and our experiments do not clearly demonstrate whether their implementation is worthwhile in all problem types. Despite these findings, IIS cuts have similar or better performance than No-Good cuts in all three problem types and can be highly beneficial on problems where the underlying structure of the instances leads to more significant differences between the IIS and No-Good cuts.

## 8 CONCLUSIONS

This work explores different strategies based on Decision Diagrams (DD) to address scheduling problems with chance constraints, specifically the Chance-Constrained Parallel Machine Scheduling Problem (CC-PMSP). We formally define the problem and present a decomposition framework where the master problem assigns jobs to machines and the subproblems checks the feasibility of the joint chance constraints. Specifically, our decomposition involves a subproblem for each machine and scenario, where we need to verify whether the assigned jobs can be sequenced within the machine's time limit.

Given that the subproblems are NP-hard due to the presence of setup times, we explore the alternative of solving these subproblems using DDs. DDs are graphical structures that can represent the feasibility set of a problem. We leverage this property

to build just a few decision diagrams (DDs) to describe the subproblems and reuse the DD graph to solve subproblems for different scenarios and machines (i.e., only changing the cost function). Moreover, we explore two DD models for these subproblems, where DD-LJ is larger but has a linear cost function, whereas DD-JS is smaller, but the cost computation is non-linear and thus more computationally expensive.

We present novel algorithms to obtain No-Good and Irreducible Infeasible Set (IIS) cuts from the DDs, which can be potentially used for other applications. We also explore the use of Benders' cuts with DDs by adapting the techniques proposed by Lozano and Smith [42], but, unfortunately, with poor performance for our specific problem. Nonetheless, we believe that such methodology could be beneficial for other applications, and we encourage researchers to consider it.

We compare our DD-based alternatives with an Integer Programming (IP) based decomposition (i.e., using IP technology to solve the subproblems and obtain No-Good or IIS cuts) across a wide range of instances coming from three different applications of the CC-PMPS (i.e., vehicle routing, operation room scheduling, and an "equal time" version of the PMSP). Our empirical results show a clear superiority of the DD-based alternatives, where our best alternative solves 55 more instances than the best IP decomposition (i.e., 184 vs. 124) and obtains significantly smaller gaps (i.e., 50% vs. 120%, on average). Our empirical evaluation presents the main findings in detail, including a discussion on why the DD-based alternatives outperform IP, a comparison of both DD models, and an analysis of using No-Good and IIS cuts.

In conclusion, this research presents a decomposition methodology based on DDs to solve scheduling problems with chance constraints, demonstrating high competitiveness. Although our work has focused specifically on the CC-PMSP, we believe that the methodology developed has significant potential to be applied to a broader range of problems. For example, the master problem could incorporate other types of constraints, focusing on entirely different domains. Likewise, the subproblems can be adapted to address other chance-constrained problems beyond scheduling, as long as they are amenable to DD technologies. This flexibility and adaptability constitute an additional contribution of our work, complementing the methodological advances already demonstrated and opening promising paths for future research at the intersection of DDs, chance constraints, and mathematical programming.

## References

1. Baker, K.R., Trietsch, D.: Principles of Sequencing and Scheduling, 2 edn. Wiley Series in Operations Research and Management Science. Wiley, Hoboken, NJ (2019)
2. Becker, B., Behle, M., Eisenbrand, F., Wimmer, R.: Bdds in a branch and cut framework. Nikoletseas, S.E. (eds) Experimental and Efficient Algorithms. WEA 2005. Lecture Notes in Computer Science **3503** (2005). DOI 10.1007/11427186_39
3. Behle, M.: Binary decision diagrams and integer programming. Ph.D. thesis, Saarland University (2007). DOI 10.22028/D291-25978
4. Bergman, D., Cire, A., van Hoeve, W.J., Hooker, J.N.: Optimization bounds from binary decision diagrams. INFORMS Journal on Computing **26**(2), 253–268 (2014). DOI 10.1287/ijoc.2013.0561
5. Bergman, D., Cire, A., van Hoeve, W.J., Hooker, J.N.: Discrete optimization with decision diagrams. INFORMS Journal on Computing **28**(1), 47 – 66 (2016). DOI 10.1287/ijoc.2015.0648
6. Bergman, D., Cire, A.A., Van Hoeve, W.J., Hooker, J.: Decision diagrams for optimization, vol. 1. Springer (2016)

7. Blazewicz, J., Ecker, K.H., Pesch, E., Schmidt, G., Sterna, M., Weglarz, J.: Handbook on Scheduling: From Theory to Practice, 2 edn. Springer Cham (2019). DOI 10.1007/978-3-319-99849-7
8. Brucker, P.: Scheduling Algorithms, 5 edn. Springer (2006)
9. Calafiore, G., Ghaoui, L.: Distributionally robust chance-constrained linear programs. J Optim Theory Appl **130**, 1–22 (2006). DOI 10.1007/s10957-006-9084-x
10. Cappart, Q., Goutierre, E., Bergman, D., Rousseau, L.M.: Improving optimization bounds using machine learning: Decision diagrams meet deep reinforcement learning. Proceedings of the AAAI Conference on Artificial Intelligence **33**(01), 1443–1451 (2019). DOI 10.1609/aaai.v33i01.33011443
11. Cardoen, B., Demeulemeester, E., Beliën, J.: Operating room planning and scheduling: A literature review. European Journal of Operational Research **201**(3), 921–932 (2010). DOI https://doi.org/10.1016/j.ejor.2009.04.011
12. Castro, M., Cire, A., Beck, C.: A combinatorial cut-and-lift procedure with an application to 0–1 second-order conic programming. Mathematical Programming **196**, 115–171 (2021). DOI 10.1007/s10107-021-01699-y
13. Castro, M., Cire, A., Beck, C.: Decision diagrams for discrete optimization: A survey of recent advances. INFORMS Journal on Computing **34**(4), 2271–2295 (2022). DOI 10.1287/ijoc.2022.1170
14. Castro, M., Piacentini, C., Cire, A., Beck, J.: Solving delete free planning with relaxed decision diagram based heuristics. Journal of Artificial Intelligence Research **67**, 607–651 (2020). DOI 10.1613/jair.1.11659
15. Castro, M.P., Bodur, M., Shalaby, A.: Incorporating service reliability in multi-depot vehicle scheduling. arXiv preprint arXiv:2407.00836 (2024)
16. Castro, M.P., Cire, A.A., Beck, J.C.: An mdd-based lagrangian approach to the multicommodity pickup-and-delivery tsp. INFORMS Journal on Computing **32**(2), 263–278 (2020)
17. Cire, A., Diamant, A., Yunes, T., Carrasco, A.: A network-based formulation for scheduling clinical rotations. Production and Operations Management **28**(5), 1186–1205 (2019). DOI https://doi.org/10.1111/poms.12978. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/poms.12978`
18. Cire, A., van Hoeve, W.J.: Multivalued decision diagrams for sequencing problems. Operations Research **61**(6), 1411–1428 (2013). DOI 10.1287/opre.2013.1221
19. Davarnia, D., van Hoeve, W.J.: Outer approximation for integer nonlinear programs via decision diagrams. Mathematical Programming **187**, 111–150 (2021). DOI 10.1007/s10107-020-01475-4
20. Deng, Y., Shen, S.: Decomposition algorithms for optimizing multi-server appointment scheduling with chance constraints. Mathematical Programming **157**, 245–276 (2016)
21. Özgün Elçi, Noyan, N.: A chance-constrained two-stage stochastic programming model for humanitarian relief network design. Transportation Research Part B: Methodological **108**, 55–83 (2018). DOI https://doi.org/10.1016/j.trb.2017.12.002. URL `https://www.sciencedirect.com/science/article/pii/S0191261517302308`
22. Excoffier, M.: Chance-constrained programming approaches for staffing and shift-scheduling problems with uncertain forecasts: application to call centers. Le moteur de recherche des thèses françaises (2015). URL `http://www.theses.fr/2015PA112244`
23. Ford, D., Fulkerson, D.: Flows in networks. The Rand Corporation (1972). URL `https://www.rand.org/content/dam/rand/pubs/reports/2007/R375.pdf`
24. Guo, C., Bodur, M., Aleman, D.M., Urbach, D.R.: Logic-based benders decomposition and binary decision diagram based approaches for stochastic distributed operating room scheduling. INFORMS Journal on Computing **33**(4), 1551–1569 (2021). DOI 10.1287/ijoc.2020.1036
25. Haus, U.U., Michini, C., Laumanns, M.: Scenario aggregation using binary decision diagrams for stochastic programs with endogenous uncertainty (2017). URL `https://arxiv.org/abs/1701.04055`
26. van Hoeve, W.J.: Graph coloring with decision diagrams. Mathematical Programming **192**, 631–674 (2022). DOI 10.1007/s10107-021-01662-x
27. van Hoeve, W.J.: An introduction to decision diagrams for optimization. INFORMS TutORials in Operations Research pp. 117–145 (2024). DOI 10.1287/educ.2024.0276
28. Hooker, J.: Discrete global optimization with binary decision diagrams. GICOLAG (2006)
29. Hu, A.J.: Techniques for efficient formal verification using binary decision diagrams. ProQuest Dissertations & Theses Global (1959)
30. Kan, A.H.G.R.: Machine Scheduling Problems, 1 edn. Springer New York, NY (1976)
31. Kinable, J., Cire, A.A., van Hoeve, W.J.: Hybrid optimization methods for time-dependent sequencing problems. European Journal of Operational Research **259**(3), 887–897 (2017)

32. Kowalczyk, D., Leus, R.: A branch-and-price algorithm for parallel machine scheduling using zdds and generic branching. INFORMS Journal on Computing **30**(4), 768–782 (2018). DOI 10.2139/ssrn. 2887850
33. Küçükyavuz, S.: Mixing sets arising in chance-constrained programming. Mathematical Programming **132**, 31–56 (2012). DOI 10.1007/s10107-010-0385-3
34. Latour, A., Babaki, B., Dries, A., Kimmig, A., den Broeck, G.V., Nijssen, S.: Combining stochastic constraint optimization and probabilistic programming. Proceedings of the International Conference on Principles and Practice of Constraint Programming pp. 495–511 (2017). DOI 10.1007/978-3-319-66158-2_32
35. Latour, A., Babaki, B., Nijssen, S.: Stochastic constraint propagation for mining probabilistic networks. Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence **8**, 1137–1145 (2019)
36. Lee, C.: Representation of switching circuits by binary-decision programs. The Bell System Technical Journal **38**(4), 985–999 (1959). DOI 10.1002/j.1538-7305.1959.tb01585.x
37. LENSTRA, J.: SEQUENCING BY ENUMERATIVE METHODS, 1 edn. MATH EMATISCH CENTRUM (1977)
38. Liu, X., Küçükyavuz, S., Luedtke, J.: Decomposition algorithms for two-stage chance-constrained programs. Mathematical Programming **157**, 219–243 (2016). DOI 10.1007/s10107-014-0832-7
39. Lodi, A., Malaguti, E., Nannicini, G., Thomopulos, D.: Nonlinear chance-constrained problems with applications to hydro scheduling. Mathematical Programming **191** (2019). DOI 10.1007/s10107-019-01447-3
40. Loekito, E., Bailey, J., Pei, J.: A binary decision diagram based approach for mining frequent subsequences. Knowl Inf Syst **24**, 235–268 (2010). DOI 10.1007/s10115-009-0252-9
41. Lozano, L., Magazine, M.J., Polak, G.G.: Decision diagram-based integer programming for the paired job scheduling problem. IISE Transactions **53**(6), 671–684 (2020). DOI 10.1080/24725854.2020.182. URL `https://ideas.repec.org/a/taf/uiiexx/v53y2020i6p671-684.html`
42. Lozano, L., Smith, J.: A binary decision diagram based algorithm for solving a class of binary two-stage stochastic programs. Mathematical Programming **191**, 381–404 (2018). DOI 10.1007/s10107-018-1315-z
43. Luedtke, J.: A branch-and-cut decomposition algorithm for solving chance-constrained mathematical programs with finite support. Mathematical Programming **146**, 219–244 (2014). DOI 10.1007/s10107-013-0684-6
44. MacNeil, M., Bodur, M.: Leveraging decision diagrams to solve two-stage stochastic programs with binary recourse and logical linking constraints. European Journal of Operational Research **315**(1), 228–241 (2024). DOI 10.1016/j.ejor.2023.12.021
45. Maschler, J., Raidl, G.: Multivalued decision diagrams for prize-collecting job sequencing with one common and multiple secondary resources. Annals of Operations Research **302**, 507–531 (2021). DOI 10.1007/s10479-019-03479-6
46. Miller, C.E., Tucker, A.W., Zemlin, R.A.: Integer programming formulation of traveling salesman problems. J. ACM **7**(4), 326–329 (1960). DOI 10.1145/321043.321046. URL `https://doi.org/10.1145/321043.321046`
47. Noorizadegan, M., Seifi, A.: An efficient computational method for large scale surgery scheduling problems with chance constraints. Computational Optimization and Applications (2018). DOI 10.1007/s10589-017-9947-0
48. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems, 6 edn. Springer Cham (2023). DOI 10.1007/978-3-031-05921-6
49. Prékopa, A.: On probabilistic constrained programmming. Proceedings of the Princeton Symposium on Mathematical Programming (1970)
50. Raghunathan, A.U., Bergman, D., Hooker, J., Serra, T., Kobori, S.: Seamless multimodal transportation scheduling (2022). URL `https://arxiv.org/abs/1807.09676`
51. Rudich, I., Cappart, Q., Rousseau, L.M.: Peel-and-bound: Generating stronger relaxed bounds with multivalued decision diagrams. 28th International Conference on Principles and Practice of Constraint Programming (CP 2022). Leibniz International Proceedings in Informatics (LIPIcs) **235**, 35:1–35:20 (2022). DOI 10.4230/LIPIcs.CP.2022.35
52. Shen, J., Zhu, Y.: Chance-constrained model for uncertain job shop scheduling problem. Soft Computing **20**, 2383–2391 (2016). DOI 10.1007/s00500-015-1647-z
53. Song, G., Leus, R.: Parallel machine scheduling under uncertainty: Models and exact algorithms. INFORMS Journal on Computing **34**(6), 3059–3079 (2022)

54. Tjandraatmadja, C., van Hoeve, W.J.: Target cuts from relaxed decision diagrams. INFORMS Journal on Computing **31**(2), 285–301 (2019). DOI 10.1287/ijoc.2018.0830
55. Wang, X., Ning, Y.: Uncertain chance-constrained programming model for project scheduling problem. Journal of the Operational Research Society (2017). DOI 10.1057/s41274-016-0122-2
56. Wegener, I.: Branching Programs and Binary Decision Diagrams: Theory and Applications. Discrete Mathematics and Applications (2000)

## A Miller-Tuckek-Zemlin Formulation

The formulation of Miller-Tucker-Zemlin (MTZ) [46], in our context, includes the following constraints:

$$\mathscr{S}_m^{MTZ}(\mathbf{y},\omega) = \{\mathbf{v}^{m\omega} \in \mathbb{R}^{|\mathscr{J}|+1} :$$

$$v_j^{m\omega} - v_k^{m\omega} + t_j^{\omega} + d_{jk}^{\omega} \leq M(1 - y_{jk}^{m\omega}) \qquad \forall j \in \mathscr{J}, k \in \mathscr{J} \cup \{0\} \tag{19a}$$

$$0 \leq v_j^{m\omega} \leq T \qquad \forall j \in \mathscr{J} \cup \{0\}\} \tag{19b}$$

The vector $\mathbf{v}^{m\omega} = (v_0^{m\omega},...,v_{|\mathscr{J}|}^{m\omega})$ functions as a time counter for the sequence of jobs assigned to machine $m$ in scenario $\omega$. Each variable $v_j^{m\omega}$ represents the moment at which job $j \in \mathscr{J}$ can begin on machine $m \in \mathscr{M}$. The variable associated with the dummy node, $v_0^{m\omega}$, indicates the total time needed to complete the entire sequence of jobs on that machine. The parameter $M$ is an upper bound on the total time required to complete any sequence of jobs.

Constraints (19b) limit the variables of vector $\mathbf{v}^{m\omega}$ within their possible values. In contrast, constraints (19a) ensure the correct assignment of values to these variables, adding the corresponding execution and setup times. If a job $j \in \mathscr{J}$ precedes job $k \in \mathscr{J}$, then the right side of constraints (19a) becomes zero and we force the time counter $v_k^{m\omega}$ to be greater than or equal to the counter $v_j^{m\omega}$ plus the corresponding execution and setup times. In case $j$ and $k$ are not together in the sequence, then we do not force any relationship between the time counters $v_j^{m\omega}$ and $v_k^{m\omega}$.

## B Deterministic equivalent model

$$\max \quad \sum_{j\in\mathscr{J}}\sum_{m\in\mathscr{M}} u_j x_{jm} \tag{DETEQ}$$

$$\text{s.t.} \quad \sum_{m\in\mathscr{M}} x_{jm} \leq 1 \qquad \forall j \in \mathscr{J} \tag{20a}$$

$$\sum_{j\in\mathscr{J}} x_{jm} \leq B \qquad \forall m \in \mathscr{M} \tag{20b}$$

$$\sum_{\omega\in\Omega} p^{\omega} z^{\omega} \geq 1 - \varepsilon \tag{20c}$$

$$\sum_{k\in\mathscr{J}\cup\{0\}} y_{jk}^{m\omega} = x_{jm} \qquad \forall j \in \mathscr{J}, m \in \mathscr{M}, \omega \in \Omega \tag{20d}$$

$$\sum_{j\in\mathscr{J}\cup\{0\}} y_{jk}^{m\omega} = x_{km} \qquad \forall k \in \mathscr{J}, m \in \mathscr{M}, \omega \in \Omega \tag{20e}$$

$$\sum_{k\in\mathscr{J}\cup\{0\}} y_{0k}^{m\omega} = 1 \qquad \forall m \in \mathscr{M}, \omega \in \Omega \tag{20f}$$

$$\sum_{j\in\mathscr{J}\cup\{0\}} y_{j0}^{m\omega} = 1 \qquad \forall m \in \mathscr{M}, \omega \in \Omega \tag{20g}$$

$$y_{jj}^{m\omega} = 0 \qquad \forall j \in \mathscr{J}, m \in \mathscr{M}, \omega \in \Omega \tag{20h}$$

In case of opting to use the MTZ formulation, the following constraints are included:

$$0 \leq v_j^{m\omega} \leq T \qquad \forall j \in \mathscr{J} \cup \{0\}, m \in \mathscr{M}, \omega \in \Omega \tag{21a}$$

$$v_j^{m\omega} - v_k^{m\omega} + t_j^{\omega} + d_{jk}^{\omega} \leq M(1 - y_{jk}^{m\omega}) \quad \forall j \in \mathscr{J}, k \in \mathscr{J} \cup \{0\}, m \in \mathscr{M}, \omega \in \Omega \tag{21b}$$

In case of wanting to use the SCF formulation, the following constraints are applied:

$$u_{jk}^{m\omega} \leq T y_{jk}^{m\omega} \qquad \begin{matrix} \forall j,k \in \mathscr{J} \cup \{0\}, \\ m \in \mathscr{M}, \omega \in \Omega \end{matrix} \tag{22a}$$

$$\sum_{k\in\mathscr{J}\cup\{0\}} u_{jk}^{m\omega} - \sum_{k\in\mathscr{J}\cup\{0\}} u_{kj}^{m\omega} = \sum_{k\in\mathscr{J}\cup\{0\}} y_{jk}^{m\omega}(t_j^{\omega} + d_{jk}^{\omega}) \qquad \begin{matrix} \forall j \in \mathscr{J}, m \in \mathscr{M}, \\ \omega \in \Omega \end{matrix} \tag{22b}$$

$$u_{jk}^{m\omega} \geq 0 \qquad\qquad \begin{aligned} &\forall j,k \in \mathscr{J}, m \in \mathscr{M}, \\ &\omega \in \Omega \end{aligned} \qquad (22c)$$

## C Symmetry Constraints Example

*Example 5 (Symmetry Constraints)* Let us take an example with 3 machines and 5 jobs, in which we have the following candidate solutions:

$$\mathbf{x}^1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \mathbf{x}^2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

We can verify that both solutions are symmetric, since in $\mathbf{x}^2$ the sets of jobs assigned to machines 1 and 3 of $\mathbf{x}^1$ were exchanged. But by adding constraints (9), $\mathbf{x}^2$ is no longer valid as it breaks both sets of constraints. Constraints (9a) are broken because $x_{1,3}^2 = 1$ and yet, the machine index is greater than the job index ($3 > 1$). But it also breaks constraints (9b), because if we take $j = 2$ and $m = 1$ we have that $x_{2,2}^2 = 1$ and that $x_{1,1} = 0$, contradicting what the constraint asks us ($x_{2,2}^2 \leq x_{1,1}$). $\qquad\square$

## D Other Improvements to Master Candidate Solutions

*Master Scenarios.* This strategy employs the first $\lfloor \varepsilon \cdot |\Omega| \rfloor + 1$ scenarios, which we will identify in the set $\Omega^{MS}$, and we add the following constraints to the master problem:

$$\sum_{\omega \in \Omega^{MS}} p^\omega z^\omega \geq 1 - \varepsilon - \sum_{\omega \in \Omega \setminus \Omega^{MS}} p^\omega \qquad (23a)$$

$$\mathscr{Q}(x, \omega, m) = 1 \qquad\qquad \forall m \in \mathscr{M}, \omega \in \Omega^{MS} \qquad (23b)$$

In constraint (23a) we seek that the candidate solution does not exceed the limit of unfulfilled scenarios in $\Omega^{MS}$. For example, consider $|\Omega| = 100$ and $\varepsilon = 0.1$, and therefore, $|\Omega^{MS}| = 100 \cdot 0.1 + 1 = 11$. The chance constraint (6c) imposes that at most 10 scenarios do not comply with the scheduling restrictions (SubIP). Constraint (23a) ensures that candidate solutions comply with (SubIP) in at least one scenario $\omega \in \Omega^{MS}$, ensuring that this solution meets the requirements of (6c) in at least the first scenarios. Constraints (23b) are the same as in (SubIP), only that variable $x$ is not fixed.

*Optimistic.* This strategy considers the most optimistic possible scenario $\omega^*$ and adds the following constraints to the master problem:

$$\mathscr{Q}(x, \omega^*, m) = 1 \qquad \forall m \in \mathscr{M} \qquad (24a)$$

The constraints are the same as in (SubIP), only that variable $x$ is not fixed and we force feasibility with $z = 1$. Additionally, it uses the optimistic scenario $\omega^*$ in which each job has an execution time of $t_j^{\omega^*} = \min\limits_{\omega \in \Omega}\{t_j^\omega\}$ and setup times of $d_{jk}^{\omega^*} = \min\limits_{\omega \in \Omega}\{d_{jk}^\omega\}$.

## E Top-Down Algorithm

The DD construction is performed directly through Algorithm 3, which employs the Top-Down construction approach [4]. The algorithm begins by initializing the DD structure, including the root and terminal nodes (lines 2-5). Subsequently, the DD structure is traversed starting from the root root node. For each node, an arc $a$ is generated corresponding to each of the values that its domain can assume (lines 7-9).

In lines 10-15, it verifies if there exists a node $n'$ with a state equivalent to the result of the transition function associated with arc $a$. If such a node exists, the arc is connected to node $n'$; otherwise, a new node is created and the arc is connected to this. Finally, in lines 16-17, the connection between the nodes of the penultimate layer and the terminal node is established, completing the DD structure.

---

**Algorithm 3** DD Top-Down Construction

---

1: **procedure** TOPDOWNCONSTRUCTION($\mathbf{x}$)
2:     Create DD $\mathscr{D} = (\mathscr{N}, \mathscr{A})$ with $|\mathbf{x}| + 1 = k + 1$ empty layers.
3:     Create the root node ($\hat{r}$) with the initial state.
4:     Create the terminal node ($\hat{t}$) with a provisionally empty state.
5:     The node $\hat{r}$ and the node $\hat{t}$ are added to layers $\mathscr{L}_1$ and $\mathscr{L}_{k+1}$, respectively.
6:     **for** $l \in \{1, ..., k\}$ **do**
7:         **for** $n \in \mathscr{L}_l$ **do**
8:             **for** $v \in \mathsf{dom}(n)$ **do**
9:                 Create an arc $a$ with $\mathsf{val}(a) = v$
10:                **if** There exists a node $n' \in \mathscr{L}_{l+1}$ with $\mathsf{s}(n') = \mathsf{tran}(\mathsf{s}(n), \mathsf{val}(a))$ **then**
11:                    Point arc $a$ to node $n'$, that is, $\mathsf{in}(a) = n'$
12:                **else**
13:                    Create node $n'$ with $\mathsf{s}(n') = \mathsf{tran}(\mathsf{s}(n), \mathsf{val}(a))$
14:                    Add node $n'$ to layer $\mathscr{L}_{l+1}$
15:                    Point arc $a$ to $n'$, this is, $\mathsf{in}(a) = n'$
16:    **for** $n \in \mathscr{L}_k$ **do**
17:        **for** $v \in \mathsf{dom}(n)$ **do**
18:            Create an arc $a$ with $\mathsf{out}(a) = n$, $\mathsf{val}(a) = v$ and $\mathsf{in}(a) = \hat{t}$
19:    **return** $\mathscr{D}$

---

## F Models Based on Network Flow

### F.1 Capacitated DD for Scheduling Problems

To solve the network flow for our scheduling problem, we will present two types of DD: Capacitated Binary Decision Diagram (BDD-CAP) and Capacitated Multiple Decision Diagram (MDD-CAP). The first is the adaptation for scheduling problems of the BDD proposed by Lozano & Smith [42], while the MDD is our contribution to reduce the size of the DD.

To adapt the capacitated arcs to our scheduling problem, two types of capacitated arcs were designed: assignment and non-assignment arcs. In the context of CC-PMSP, the requirements of these arcs relate to the assignment of jobs to a machine to allow flow in the diagram. In the case of assignment arcs, a specific job is required to be assigned to the machine for the flow to use that arc. On the other hand, in non-assignment arcs, one or more jobs are required not to be assigned to the machine to allow the flow. These requirements that each arc $a \in \mathscr{A}$ has to allow the flow will be identified with the set $\mathscr{U}_a$.

It is worth noting that, to use the capacitated arcs, the construction of both DDs used in this model includes all the jobs to be optimized, i.e., $\mathscr{J}$. This is because the capacitated arcs evaluate whether the assignment or non-assignment of a job to a machine could improve the result, so they need to evaluate all jobs. However, this could represent a disadvantage compared to the previously presented models, DD-LJ and DD-JS, which use only the jobs assigned to a certain machine, thus limiting the size of the layers to $|\mathscr{L}| \leq B$. In contrast, the layers of the DDs described in this section are bounded by $|\mathscr{J}|$, which implies a larger size.

#### F.1.1 BDD-CAP Model

This version of the model employs the BDD presented by Lozano & Smith [42], whose structure differs from those used in DD-LJ and DD-JS. In this model, a decision vector $\mathbf{q} \in \{0,1\}^{|\mathscr{J}^m| \cdot |\mathscr{J}^m|}$ is defined, where each variable $q_{jp}$ indicates whether job $j$ has been assigned to position $p$ within the sequence. Consequently, for this type of DD, each layer $\mathscr{L}_{jp} \in \mathscr{L}$ is related to the variable $q_{jp}$, and for all nodes $n \in \mathscr{N}$, $\mathsf{dom}(n) = \{0,1\}$. The value 1 indicates that job $j$ is assigned to position $p$, while the value 0 indicates that it will not be assigned to that position.

To construct the BDD-CAP, the same composite states defined in (12) for DD-LJ are employed, as well as the same initial value $(\{\}, -1)$. However, the transition function differs, since in this case $\mathsf{val}(a) \in \{0,1\}$ for all $a \in \mathscr{A}$. This implies that the values of the arcs no longer represent the job $j \in \mathscr{J}$ to which a

position is to be assigned, but rather reflect a binary decision. The transition function of an arc $a = (n, n') \in \mathscr{A}$, where $n \in \mathscr{L}_{jp}$, is defined as:

$$\mathsf{tran}(\mathsf{s}(n), \mathsf{val}(a)) := \begin{cases} (\mathsf{s}_{set}(n) \cup \{j\}, j) & \text{if } \mathsf{val}(a) = 1 \\ (\mathsf{s}_{set}(n), \mathsf{s}_{last}(n)) & \text{if } \mathsf{val}(a) = 0 \end{cases} \quad (25)$$

It is important to note that, in case $\mathsf{val}(a) = 1$, the job $j$ is used, which corresponds to the first index of the layer to which node $n$ belongs. On the other hand, when $\mathsf{val}(a) = 0$, the state of the resulting node $n'$ is the same as the state of the parent node $n$, i.e., $\mathsf{s}(n) = \mathsf{s}(n')$.

We can differentiate three types of arcs in these DDs: non-capacitated arcs, assignment capacitated arcs, and non-assignment capacitated arcs. The sets of assignment capacitated arcs and non-assignment capacitated arcs will be identified with $\mathscr{A}_1$ and $\mathscr{A}_0$, respectively. The arcs that assign a job $j \in \mathscr{J}$ to position $p \in \{1, ..., |\mathscr{J}|\}$, i.e., those arcs $a \in \mathscr{A}$ for which $\mathsf{val}(a) = 1$ are assignment capacitated arcs and require that job $j$ be assigned to the machine. On the other hand, arcs that decide not to assign job $j$ to position $p$ and are directed to another node that is not the terminal, i.e., arcs $a \in \mathscr{A}$ with $\mathsf{in}(a) \neq \hat{t}$ are non-capacitated arcs. And finally, arcs that do not assign job $j$ to position $p$ and are directed to the terminal node, $(\mathsf{in}(a) = \hat{t})$ are non-assignment capacitated arcs and require that the jobs not assigned to a position are not assigned to the machine. These last arcs jump directly to the terminal node and are the only arcs that can skip layers.

The cost of each arc $a = (n, n')$ depends on whether it is an assignment arc, a non-assignment arc, or if it is not a capacitated arc, and is given by:

$$\mathsf{c}(a, \omega) = \begin{cases} d^{\omega}_{\mathsf{val}(a),0} & \text{if } a \in \mathscr{A}_0 \\ t^{\omega}_{\mathsf{val}(a)} & \text{if } a \in \mathscr{A}_1 \text{ and } n' = \hat{r} \\ d^{\omega}_{\mathsf{s}_{last}(\mathsf{out}(a)),\mathsf{val}(a)} + t^{\omega}_{\mathsf{val}(a)} + d^{\omega}_{\mathsf{val}(a),0} & \text{if } a \in \mathscr{A}_1 \text{ and } n = \hat{t} \\ d^{\omega}_{\mathsf{s}_{last}(\mathsf{out}(a)),\mathsf{val}(a)} + t^{\omega}_{\mathsf{val}(a)} & \text{if } a \in \mathscr{A}_1 \\ 0 & \text{if } a \notin \mathscr{A}_0 \cup \mathscr{A}_1 \end{cases} \quad (26)$$

We can see that the non-capacitated arcs have no cost. The non-assignment capacitated arcs, as they go directly to the terminal node, only add the setup time for node $\hat{t}$. Finally, the assignment capacitated arcs have the same logic explained in (14) in which the execution time of the job and the corresponding setup times are added.

*Example 6 (BDD-CAP)* Let us consider that we want to assign three jobs $\mathscr{J} = \{1, 2, 3\}$ to two different machines. In Figure 7, we can see the BDD-CAP that would be used to solve this problem. It is important to note that both subproblems associated with each machine would use the same BDD-CAP, which includes all jobs.

This time the DD has nine layers, since each of the three jobs can be in any of the three positions. Each of these layers $\mathscr{L}_{jp} \in \mathscr{L}$ is associated with the variable $q_{jp}$ and they decide whether job $j$ goes in position $p$ or not. The continuous arcs are those with $\mathsf{val}(a) = 1$, while the discontinuous arcs are those with $\mathsf{val}(a) = 0$. The black arcs are non-capacitated arcs, the green arcs are assignment capacitated arcs, and the red arcs are non-assignment capacitated arcs. The capacitated arcs have a set over the arrow that represents the set $\mathscr{U}_a$, that is, they represent their capacity. This set has the jobs that must be assigned or not assigned to the machine, depending on whether it is an assignment or non-assignment arc, for the flow to be able to pass through the arc.

For example, let us consider the node that is in the center of the DD with state $(\{2\}, 2)$ that is just before assigning a value to the variable $q_{32}$, i.e., the DD must decide whether to include job 3 in the second position of the sequence. From this node, two arcs emerge: one assignment capacitated (green) and another non-assignment capacitated (red). For the flow to traverse the green arc, since $\mathscr{U}_a = \{3\}$, job 3 must be assigned to the machine. Therefore, $\hat{x}$ must be equal to $(1, 1, 1)$ or $(0, 1, 1)$ since job 2 has already been assigned. On the other hand, since the red arc is a non-assignment capacitated arc with $\mathscr{U}_a = \{1, 3\}$, jobs 1 and 3 must not be assigned to the machine to allow the flow to pass through that arc, i.e., $\hat{x} = (0, 1, 0)$. Finally, the non-capacitated arcs always allow the flow to pass. □
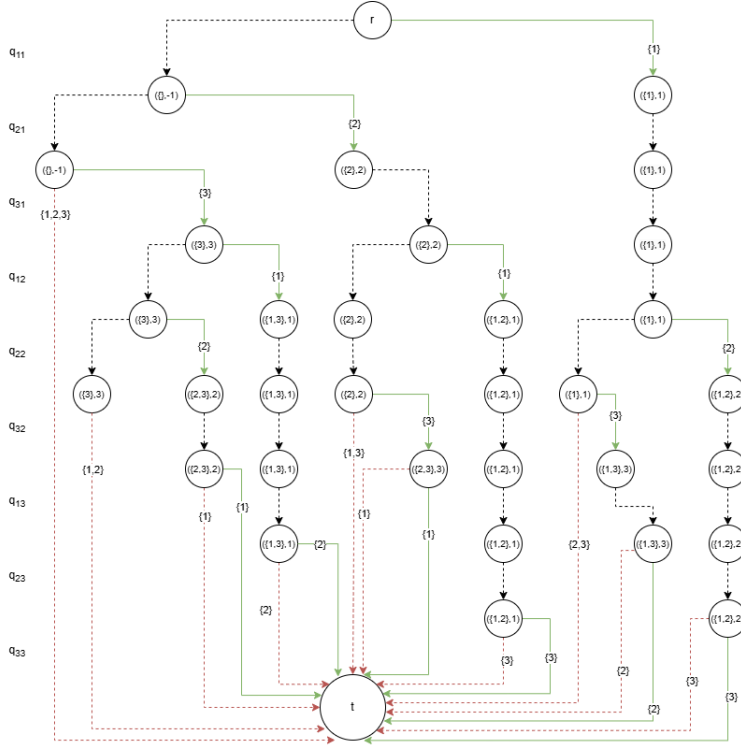
**Fig. 7** BDD-CAP structure for a subproblem with three jobs assigned to a machine

### F.1.2 MDD-CAP Model

The version of the model that uses an MDD-CAP is an adaptation of the BDD used by Lozano & Smith [42] and which we propose with the intention of reducing the size of the DD. The structure of the diagram is similar to the DDs proposed in Section 5, where the decision variables are the same as those used by DD-LJ and DD-JS. Specifically, we consider a vector $\mathbf{w} \in \mathbb{Z}_+^{|\mathscr{J}|}$ that assigns each job a position in the sequence, where the value of the variable $w_p$ indicates the index of the job assigned to position $p \in \{1, ..., |\mathscr{J}|\}$ or takes the value $-1$ in case it is a non-assignment arc. Therefore, for each node $n \in \mathscr{N}$, the domain of the associated variable is $\mathsf{dom}(n) = (\mathscr{J} \cup \{-1\}) \setminus \mathsf{s}_{set}(n)$.

It is worth noting that for this DD, the vector $\mathbf{w}$ is larger and the values of $p$ are bounded by the complete set of jobs and not by the set of jobs assigned to a machine $m \in \mathscr{M}$. This results in the MDD-CAP used in this model being considerably larger than the DD-LJ and DD-JS.

In this case, there are only two types of arcs and both are capacitated: the assignment arcs and the non-assignment arcs. These sets of arcs will maintain the same notation used in the BDD-CAP, i.e., $\mathscr{A}_1$ and $\mathscr{A}_0$, respectively. If the arc $a \in \mathscr{A}$ has $\mathsf{val}(a) \neq -1$, then it is an assignment arc. On the contrary, if the arc $a$ has $\mathsf{val}(a) = -1$, then it is a non-assignment arc.

To construct the MDD-CAP, the same states as in DD-LJ and BDD-CAP are used, defined in (12), and its initial value is also $(\{\}, -1)$. The transition function is similar to that presented in (25), only that the arcs can take more values.

$$\mathsf{tran}(\mathsf{s}(n), \mathsf{val}(a)) := \begin{cases} (\mathsf{s}_{set}(n) \cup \{\mathsf{val}(a)\}, \mathsf{val}(a)) & \text{if } \mathsf{val}(a) \neq -1 \\ (\mathsf{s}_{set}(n), \mathsf{s}_{last}(n)) & \text{if } \mathsf{val}(a) = -1 \end{cases} \qquad (27)$$

The cost function is also similar to that presented in (26) for the BDD-CAP. We have that for all arcs $a = (n, n')$ and $\omega \in \Omega$:

$$\mathsf{c}(a, \omega) = \begin{cases} d^{\omega}_{\mathsf{val}(a), \hat{t}} & \text{if } a \in \mathscr{A}_0 \\ t^{\omega}_{\mathsf{val}(a)} & \text{if } a \in \mathscr{A}_1 \text{ and } n' = \hat{r} \\ d^{\omega}_{\mathsf{s}_{last}(\mathsf{out}(a)), \mathsf{val}(a)} + t^{\omega}_{\mathsf{val}(a)} + d^{\omega}_{\mathsf{val}(a), 0} & \text{if } a \in \mathscr{A}_1 \text{ and } n = \hat{t} \\ d^{\omega}_{\mathsf{s}_{last}(\mathsf{out}(a)), \mathsf{val}(a)} + t^{\omega}_{\mathsf{val}(a)} & \text{if } a \in \mathscr{A}_1 \end{cases} \tag{28}$$

For the non-assignment capacitated arcs, the setup time for node $\hat{t}$ is added, since they always go to the terminal node. For the assignment capacitated arcs, the execution time, the corresponding setup time in case it is not the first assigned job, and the setup time towards $\hat{t}$ in case it is directed to the terminal node, must always be added.

*Example 7 (MDD-CAP)* Let us consider the same set of jobs as in Example 6. In Figure 8, we can see the MDD-CAP that would be used to solve this problem. The MDD-CAP in the figure has three layers, one for each of the positions to which a job can be assigned, and each of these layers is associated with a variable $w_p$.

The continuous arcs are assignment arcs, while the discontinuous arcs are non-assignment arcs. Each of the colors of the arcs represents its value: black is $\mathsf{val}(a) = 1$, green is $\mathsf{val}(a) = 2$, purple is $\mathsf{val}(a) = 3$, and red is $\mathsf{val}(a) = -1$. The set over each of the arcs represents the set $\mathscr{U}_a$.

Let us consider the candidate solution $\hat{\mathbf{x}} = (1, 0, 1)$. From the root node, it is only possible to take the black and purple arcs, since the green arc is an assignment capacitated arc with capacity $\mathscr{U}_a = \{2\}$ and job 2 is not part of the solution.

From the node with state $(\{1\}, 1)$, the only available arc is the purple one. The green arc is still inaccessible for the same reason mentioned in the root node, while the red arc is a non-assignment capacitated arc with $\mathscr{U}_a = \{2, 3\}$. However, since job 3 is assigned to the machine, the flow cannot transit through that arc.

Similarly, from the node with state $(\{3\}, 3)$, it is only possible to take the black arc. The non-assignment capacitated arc prevents the flow from passing, since its capacity includes job 1, which is assigned to the machine. In the same way, the assignment capacitated arc is not traversable due to its capacity including job 2.

Finally, in both paths, we arrive at a node with state $(\{1, 3\}, 1)$ and another with state $(\{1, 3\}, 3)$. In each case, there are two available arcs: one assignment capacitated and another non-assignment capacitated, both with $\mathscr{U}_a = \{2\}$. Since job 2 is not part of the solution, the only traversable arc is the red one. □
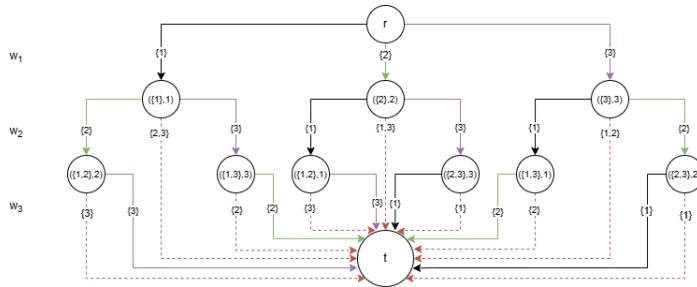


**Fig. 8** MDD-CAP structure for a subproblem with three jobs assigned to a machine

## F.2 Network Flow and Cuts

Given one of the DDs described in the previous section, the network flow problem to solve our subproblem has the following form for a machine $m \in \mathcal{M}$ and scenario $\omega \in \Omega$:

$$\min \quad \sum_{a \in \mathscr{A}} \mathsf{c}(a, \omega) \cdot v_a \qquad\qquad (\text{NetworkFlow})$$

$$\text{s.t.} \sum_{a \in \delta^{out}(n)} v_a - \sum_{a \in \delta^{in}(n)} v_a = \begin{cases} 1 & n = \hat{r} \\ 0 & \forall n \in \mathscr{N} \setminus \{\hat{r}, \hat{t}\} \\ -1 & n = \hat{t} \end{cases} \qquad (29a)$$

$$v_a \leq \hat{x}_{qm} \qquad \forall a \in \mathscr{A}_1, q \in \mathscr{U}_a \qquad (29b)$$

$$v_a \leq 1 - \hat{x}_{qm} \qquad \forall a \in \mathscr{A}_0, q \in \mathscr{U}_a \qquad (29c)$$

$$v_a \geq 0 \qquad \forall a \in \mathscr{A} \qquad (29d)$$

The objective function seeks to minimize the times associated with the flow by passing from the root node $\hat{r}$ to the terminal node $\hat{t}$ in scenario $\omega$. The constraints defined in (29a) guarantee the logical consistency of the flow in the network and that the flow starts at $\hat{r}$ and ends at $\hat{t}$.

The set $\mathscr{U}_a$ contains the capacities of arc $a \in \mathscr{A}$, that is, it contains the jobs required for the flow to pass through the arc. In case $a \in \mathscr{A}_1$, this requires that the jobs $q \in \mathscr{U}_a$ be assigned to the machine. On the other hand, if $a \in \mathscr{A}_0$, the arc requires that the jobs $q \in \mathscr{U}_a$ not be assigned to the machine. Therefore, constraints (29b) and (29c) impose on the model to comply with the requirements of the assignment arcs and the non-assignment arcs. If a job $q \in \mathscr{U}_a$ of the capacity of arc $a \in \mathscr{A}_1$ is not assigned to machine $m \in \mathscr{M}$, i.e., $\hat{x}_{qm} = 0$, then in the corresponding constraints of (29b) forces $v_a \leq 0$, preventing the flow from passing through that arc. Similarly, for an arc $a \in \mathscr{A}_0$, if a job $q \in \mathscr{U}_a$ is assigned to machine $m$, i.e., $\hat{x}_{qm} = 1$, then in the corresponding constraint of (29c) forces $v_a \leq 1 - 1 = 0$, blocking the flow through the arc.

If we define $\pi$, $\alpha$ and $\beta$ as the dual variables of (29a), (29b) and (29c), respectively, the dual problem is of the following form:

$$\max \quad \pi_{\hat{r}} - \pi_{\hat{t}} + \sum_{a \in A_1} \sum_{q \in \mathscr{U}_a} \hat{x}_{qm} \cdot \alpha_{aq} + \sum_{a \in A_0} \sum_{q \in \mathscr{U}_a} (1 - \hat{x}_{qm}) \cdot \beta_{aq} \qquad (\text{DualNetworkFlow})$$

$$\text{s.t.} \quad \pi_n - \pi_{n'} + \sum_{q \in \mathscr{U}_a} \alpha_{aq} \leq \mathsf{c}(a, \omega) \qquad \forall a = (n, n') \in \mathscr{A}_1 \qquad (30a)$$

$$\pi_n - \pi_{n'} + \sum_{q \in \mathscr{U}_a} \beta_{aq} \leq \mathsf{c}(a, \omega) \qquad \forall a = (n, n') \in \mathscr{A}_0 \qquad (30b)$$

$$\alpha \leq 0 \qquad (30c)$$

$$\beta \leq 0 \qquad (30d)$$

Lozano & Smith [42] importantly note that, in general, there exist multiple valid dual solutions in (DualNetworkFlow), which can affect the Benders cut in different ways. For this reason, they impose a specific interpretation of the dual variables that proved effective in their research. In their formulation, the optimal value $\hat{\pi}_n$ of the dual variable $\pi_n$ corresponds to the shortest path from node $n$ to the terminal node $\hat{t}$, which implies that $\hat{\pi}_{\hat{t}} = 0$. The optimal values $\hat{\alpha}_{aq}$ and $\hat{\beta}_{aq}$ of the dual variables $\alpha_{aq}$ and $\beta_{aq}$ represent reductions in the time of $\hat{\pi}_{\hat{r}}$ when the capacity $q \in \mathscr{U}_a$ of arc $a \in \mathscr{A}_1$ or $a \in \mathscr{A}_0$, respectively, is satisfied. Therefore, if the arc $a \in \mathscr{A}$ is included in $\hat{\pi}_{\hat{r}}$ or if its incorporation increases the time of the path, then it is fulfilled that $\hat{\alpha}_{aq} = 0$ or $\hat{\beta}_{aq} = 0$, depending on whether it is an assignment capacitated arc or a non-assignment arc.

To solve the master problem, we use optimality cuts of a Benders decomposition but adapted to take into account the chance constraint. Therefore, the cut we propose has the following form:

$$M(z^{\omega} - 1) + \hat{\pi}_{\hat{r}} + \sum_{a \in \mathscr{A}_1} \sum_{q \in \mathscr{U}_a} \hat{\alpha}_{aq} \cdot x_{qm} + \sum_{a \in \mathscr{A}_0} \sum_{q \in \mathscr{U}_a} \hat{\beta}_{aq} \cdot (1 - x_{qm}) \leq T \qquad (31)$$

As mentioned earlier, in the cut we have that $\hat{\pi}_{\hat{r}}$ represents the time of the shortest path in the current solution, while the terms $\hat{\alpha}_{aq}$ and $\hat{\beta}_{aq}$ correspond to possible reductions in that duration due to arc $a \in \mathscr{A}$, in case of adding job $q \in \mathscr{J}$ to machine $m \in \mathscr{M}$. If the sum of these terms, $\hat{\pi}_{\hat{r}} + \sum_{a \in \mathscr{A}_1} \sum_{q \in \mathscr{U}_a} \hat{\alpha}_{aq} \cdot x_{qm} +$

$\sum_{a \in \mathscr{A}_0} \sum_{q \in \mathscr{U}_a} \hat{\beta}_{aq} \cdot (1 - x_{qm})$, is greater than $T$, then the time availability of the machine for that scenario is not met and therefore $z^{\omega} = 0$. This activates the parameter $M$ which is a sufficiently large number to omit the constraint. Otherwise, if the time availability is met, the use of parameter $M$ is not necessary since the inequality is fulfilled and consequently, $z^{\omega}$ can take the value of 1.

### F.2.1 Improvements to the Cut

As mentioned at the beginning of the section, Lozano & Smith [42] propose two strategies to improve cut (31). The first strategy takes advantage of the fact that in each layer only one arc can be taken. Therefore, adding all the decreases of all the arcs of a layer that benefit from adding a job is not a good bound. To adjust the bound, they define disjoint sets of arcs $\Gamma^1, ..., \Gamma^D$ such that $\mathscr{A}_1 = \bigcup_{d=1}^{D} \Gamma^d$ and $\Gamma^k \cap \Gamma^l = \emptyset$ for all $k, l \in \{1, ..., D\} : k \neq l$. In these sets, every possible path from $\hat{r}$ to $\hat{t}$ uses at most one arc belonging to $\Gamma^d$ for all $d \in \{1, ..., D\}$. The arcs that leave each layer in the DDs of our problem fulfill the definitions of $\Gamma$, so $D = |\mathscr{L}| - 1$. Applied to our case, the cut would be as follows:

$$M(z^{\omega} - 1) + \hat{\pi}_{\hat{r}} + \sum_{q=1}^{|\mathscr{J}|} \sum_{d=1}^{D} \hat{\gamma}_{qd} \cdot x_{qr} + \sum_{q=1}^{|\mathscr{J}|} \hat{\delta}_q \cdot (1 - x_{qr}) \leq T \tag{32}$$

where $\hat{\gamma}_{qd} = \min_{a \in \Gamma^d} \{\hat{\alpha}_{aq} | q \in \mathscr{U}_a\}$ and $\hat{\delta}_q = \min_{a \in \mathscr{A}_0} \{\hat{\beta}_{aq} | q \in \mathscr{U}_a\}$. Therefore, this time we have the minimum time of the current solution ($\hat{\pi}_{\hat{r}}$) to which possibly at most the greatest reduction in time ($\hat{\gamma}_{qd}$) of each job $q \in \mathscr{J}$ in each layer $d \in \{1, ..., D\}$ can be subtracted. Similarly, the same happens with $\hat{\delta}_q$ but with the non-assignment capacitated arcs.

This bound is tighter than the previous one because not all possible reductions of $\alpha$ and $\beta$ are taken into account, but only the minimum of them, that is, the one that can produce the greatest reduction in $\hat{\pi}_{\hat{r}}$. Note that $\delta$ does not need the sum of the disjoint sets of arcs because the non-assignment capacitated arcs all go to node $\hat{t}$ and, therefore, $\mathscr{A}_0$ is a set of disjoint arcs since no path from $\hat{r}$ to $\hat{t}$ can take two of these arcs.

The second strategy proposed by Lozano & Smith [42] consists of precomputing the optimal path for the network flow problem by forcing the flow to pass through each arc $\hat{a} \in \mathscr{A}$ such that $\mathscr{U}_{\hat{a}} \neq \emptyset$:

$$l_{\hat{a}}^{\omega} = \min \qquad \sum_{a \in \mathscr{A}_1} \mathsf{c}(a, \omega) \cdot v_a$$

$$\text{s.t.} \qquad \sum_{i \in \mathscr{J}} x_i \leq B \tag{33a}$$

$$\sum_{a \in \delta^{out}(n)} v_a - \sum_{a \in \delta^{in}(n)} v_a = \begin{cases} 1 & n = \hat{r} \\ 0 & \forall n \in \mathscr{N} \setminus \{\hat{r}, \hat{t}\} \\ -1 & n = \hat{t} \end{cases} \tag{33b}$$

$$v_a \leq x_q \qquad \forall a \in \mathscr{A}_1, q \in \mathscr{U}_a \tag{33c}$$

$$v_a \leq 1 - x_q \qquad \forall a \in \mathscr{A}_0, q \in \mathscr{U}_a \tag{33d}$$

$$v_{\hat{a}} = 1 \tag{33e}$$

$$v_a \geq 0 \qquad \forall a \in \mathscr{A} \tag{33f}$$

$$\mathbf{x} \in \{0, 1\}^{|\mathscr{J}|} \tag{33g}$$

In this way, we obtain a new lower bound, since it represents the minimum possible time when using arc $a$. This allows us to adjust the bound in a better way. The resulting cut is as follows:

$$M(z^{\omega} - 1) + \hat{\pi}_{\hat{r}} + \sum_{q=1}^{|\mathscr{J}|} \sum_{d=1}^{D} \underline{\hat{\gamma}}_{qd}^{\omega} \cdot x_{qr} + \sum_{q=1}^{|\mathscr{J}|} \underline{\hat{\delta}}_q^{\omega} \cdot (1 - x_{qr}) \leq T \tag{34}$$

where

$$\underline{\hat{\gamma}}_{qd}^{\omega} = \min_{a \in \Gamma^d} \{\underline{\hat{\alpha}}_{aq}^{\omega} \mid q \in \mathscr{U}_a\}, \qquad \underline{\hat{\alpha}}_{aq}^{\omega} = \max\{\hat{\alpha}_{aq}, (l_a^{\omega} - \hat{\pi}_{\hat{r}})^-\},$$

$$\underline{\hat{\delta}}_q^{\omega} = \min_{a \in \mathscr{A}_0} \{\underline{\hat{\beta}}_{aq}^{\omega} \mid q \in \mathscr{U}_a\}, \qquad \underline{\hat{\beta}}_{aq}^{\omega} = \max\{\hat{\beta}_{aq}, (l_a^{\omega} - \hat{\pi}_{\hat{r}})^-\}.$$

**Table 5** Summary of Unfiltered Network Flow Results

| Model | Strategy | TotalTime | Gap | #Optimal |
|---|---|---|---|---|
| BDD-CAP | - | 645.1 | inf | 45 |
| BDD-CAP | 1 | 629.0 | inf | 45 |
| MDD-CAP | - | 595.6 | inf | 71 |
| MDD-CAP | 1 | 605.6 | inf | 68 |

**Table 6** Summary of Filtered Network Flow Results

| Model | Strategy | OptimalTimes | GapWithoutInf | #Inf |
|---|---|---|---|---|
| BDD-CAP | - | 79.0 | - | 45 |
| BDD-CAP | 1 | 70.6 | - | 44 |
| MDD-CAP | - | 439.1 | 0.1 | 5 |
| MDD-CAP | 1 | 403.6 | 0.1 | 6 |

The intuition behind this cut is to take advantage of the best lower bounds obtained in the previous strategies and incorporate them into the formulation. It is important to note that the coefficients $\hat{\gamma}$ and $\hat{\underline{\delta}}$ are obtained through a minimization of the maximum between the lower bounds, thus guaranteeing the validity of the inequality. In particular, the terms $\hat{\underline{\alpha}}$ and $\hat{\beta}$ adjust the lower bound, and the minimization ensures that the constraint remains valid in all cases.

## F.3 Results of the Network Flow Model

The reason why the results of the Network Flow Model are not included in the previous results is that it was not possible to create decision diagrams with the dimensions required by these models. Unlike the decision diagrams of the MDD Model, which have a depth equal to the number of jobs assigned to the machine, the decision diagrams of the Network Flow Model have a depth equal to the total number of jobs of the complete problem. Therefore, the depth can be up to $|\mathcal{M}|$ times greater.

If we calculate the nodes of the decision diagrams of the DD-LJ model, which will always have equal or greater quantity than the nodes of the DD-JS, we find that the maximum number of nodes is $B!$. However, when calculating the number of nodes of the Network Flow Models, we obtain that they are, at most, $|\mathcal{J}|!$ for the MDD-CAP and $2^{|\mathcal{J}|\cdot|\mathcal{J}|}$ for the BDD-CAP. Therefore, the difference in sizes between the decision diagrams is considerable.

For these models, we had to lower the number of scenarios to $|\Omega| = 50$ and lower the number of jobs and machines $(|\mathcal{J}|, |\mathcal{M}|) \in \{(6, 2), (8, 2)\}$. The rest of the parameters remain exactly the same.

Tables 5 and 6 have the same columns as Tables 1 and 2. The only difference is that, as the Network Flow models do not have No-Good and IIS cuts, that column was replaced by the strategies proposed by Lozano & Smith [42]. It is important to mention that under the second strategy in these models, no results could be seen even by lowering the parameters to the minimum because the optimization of $l_{\hat{a}}^{\omega}$ for all arcs took more than the 20-minute time limit.

As can be observed in the tables, despite the considerable reduction in the number of jobs, machines, and scenarios, these models show limited performance. The MDD-CAP model manages to find 71 optimal solutions without a specific strategy and 68 with strategy 1, while the BDD-CAP reaches only 45 optimal solutions in both cases, out of a total of 90 instances evaluated for each configuration.

Table 6 reveals that the average time to reach optimal solutions is significantly lower for BDD-CAP (79.0 and 70.6 seconds) compared to MDD-CAP (439.1 and 403.6 seconds). However, the BDD-CAP model failed to find feasible solutions in 45 and 44 instances, respectively, which represents practically 50% of the cases. In contrast, MDD-CAP only failed to find feasible solutions in 5 and 6 instances, respectively, also achieving an average gap of 0.1% for sub-optimal solutions.

When contrasting these results with those obtained by the DD-JS and DD-LJ models presented earlier, the difference is notable. While the network flow models required drastically reducing the size of the

instances to just 6-8 jobs and 2 machines with 50 scenarios, the DD-JS and DD-LJ models were able to address problems with up to 140 jobs, 10 machines, and 100 scenarios. According to Table 1, DD-JS managed to find 184 and 173 optimal solutions (with IIS and No-Good cuts respectively), while DD-LJ reached 160 optimal solutions with both types of cuts, out of a total of 243 instances per configuration.

# G Experiment Results by data sets

**Table 7** Results by data sets

| Model | Cut | Dataset | #Problems | #Optimal | TotalTime |
|-------|-----|---------|-----------|----------|-----------|
| DD-JS | IIS | Equal | 135 | 84 | 709.9 |
| DD-JS | IIS | ORS | 135 | 21 | 1080.3 |
| DD-JS | IIS | VRP | 135 | 79 | 652.4 |
| DD-JS | NoGood | Equal | 135 | 72 | 758.8 |
| DD-JS | NoGood | ORS | 135 | 21 | 1090.1 |
| DD-JS | NoGood | VRP | 135 | 80 | 657.1 |
| DD-LJ | IIS | Equal | 135 | 67 | 825.2 |
| DD-LJ | IIS | ORS | 135 | 18 | 1109.6 |
| DD-LJ | IIS | VRP | 135 | 75 | 754.6 |
| DD-LJ | NoGood | Equal | 135 | 63 | 844.6 |
| DD-LJ | NoGood | ORS | 135 | 19 | 1122.7 |
| DD-LJ | NoGood | VRP | 135 | 78 | 740.3 |
| IP | IIS | Equal | 135 | 17 | 1121.1 |
| IP | IIS | ORS | 135 | 1 | 1237.5 |
| IP | IIS | VRP | 135 | 46 | 981.1 |
| IP | NoGood | Equal | 135 | 49 | 957.6 |
| IP | NoGood | ORS | 135 | 14 | 1132.7 |
| IP | NoGood | VRP | 135 | 66 | 813.9 |

## G.1 VRP

**Table 8** Summary of VRP Unfiltered Results

| Model | Cut | #Problems | #Optimal | TotalTime | Gap |
|-------|-----|-----------|----------|-----------|-----|
| DD-JS | IIS | 135 | 79 | 652.3986 | inf |
| DD-JS | No-Good | 135 | 80 | 657.1437 | 0.4160 |
| DD-LJ | IIS | 135 | 75 | 754.5817 | inf |
| DD-LJ | No-Good | 135 | 78 | 740.2876 | 0.4813 |
| IP | IIS | 135 | 46 | 981.0909 | inf |
| IP | No-Good | 135 | 66 | 813.8522 | 0.8578 |

**Table 9** Summary of VRP Filtered Results

| Model | Cut | OptimalTimes | GapWithoutInf | #Inf |
|-------|-----|-------------|---------------|------|
| DD-JS | IIS | 263.8 | 0.4 | 1 |
| DD-JS | No-Good | 283.5 | 0.4 | 0 |
| DD-LJ | IIS | 397.8 | 0.4 | 1 |
| DD-LJ | No-Good | 393.0 | 0.5 | 0 |
| IP | IIS | 448.9 | 0.7 | 70 |
| IP | No-Good | 408.5 | 0.9 | 0 |

**Table 10** Detailed VRP Results

| Model | Cut | #Callback | #Cuts | ResolTime | ResolTimePerCB |
|-------|-----|-----------|-------|-----------|----------------|
| DD-JS | IIS | 97.9 | 28134.4 | 500.1 | 5.1 |
| DD-JS | No-Good | 99.3 | 32223.9 | 504.1 | 5.1 |
| DD-LJ | IIS | 78.0 | 21939.0 | 656.0 | 8.4 |
| DD-LJ | No-Good | 88.4 | 29043.5 | 601.0 | 6.8 |
| IP | IIS | 3.5 | 99.0 | 77.6 | 22.3 |
| IP | No-Good | 33.7 | 5847.4 | 527.3 | 15.7 |

| Model | Cut | CreateCutTime | CreateSPTime | UpdateTime | ResolTime |
|-------|-----|---------------|--------------|------------|-----------|
| DD-JS | IIS | 0.7 | 0.3 | N/A | 500.1 |
| DD-JS | No-Good | 0.9 | 0.2 | N/A | 504.1 |
| DD-LJ | IIS | 0.6 | 1.4 | N/A | 656.0 |
| DD-LJ | No-Good | 0.8 | 1.4 | N/A | 601.0 |
| IP | IIS | 613.5 | 241.4 | 23.4 | 77.6 |
| IP | No-Good | 0.2 | 179.2 | 92.7 | 527.3 |

## G.2 ORS

**Table 11** Summary of ORS Unfiltered Results

| Model | Cut | #Problems | #Optimal | TotalTime | Gap |
|-------|-----|-----------|----------|-----------|-----|
| DD-JS | IIS | 135 | 21 | 1080.3 | 0.8 |
| DD-JS | No-Good | 135 | 21 | 1090.1 | 0.8 |
| DD-LJ | IIS | 135 | 18 | 1109.6 | 1.1 |
| DD-LJ | No-Good | 135 | 19 | 1122.7 | 1.0 |
| IP | IIS | 135 | 1 | 1237.5 | inf |
| IP | No-Good | 135 | 14 | 1132.7 | inf |

**Table 12** Summary of ORS Filtered Results

| Model | Cut | OptimalTimes | GapWithoutInf | #Inf |
|---|---|---|---|---|
| DD-JS | IIS | 427.1 | 0.8 | 0 |
| DD-JS | No-Good | 490.0 | 0.8 | 0 |
| DD-LJ | IIS | 518.0 | 1.1 | 0 |
| DD-LJ | No-Good | 547.3 | 1.0 | 0 |
| IP | IIS | 136.7 | 1.6 | 101 |
| IP | No-Good | 537.0 | 1.5 | 2 |

**Table 13** Detailed ORS Results

| Model | Cut | #Callback | #Cuts | ResolTime | ResolTimePerCB |
|---|---|---|---|---|---|
| DD-JS | IIS | 112.4 | 18681.2 | 919.3 | 8.2 |
| DD-JS | No-Good | 116.6 | 16235.7 | 925.4 | 7.9 |
| DD-LJ | IIS | 88.0 | 14245.7 | 1005.4 | 11.4 |
| DD-LJ | No-Good | 100.7 | 14025.8 | 976.9 | 9.7 |
| IP | IIS | 4.6 | 189.7 | 48.2 | 10.5 |
| IP | No-Good | 52.2 | 4190.3 | 779.3 | 14.9 |

| Model | Cut | CreateCutTime | CreateSPTime | UpdateTime | ResolTime |
|---|---|---|---|---|---|
| DD-JS | IIS | 0.5 | 0.4 | N/A | 919.3 |
| DD-JS | No-Good | 0.5 | 0.3 | N/A | 925.4 |
| DD-LJ | IIS | 0.4 | 1.9 | N/A | 1005.4 |
| DD-LJ | No-Good | 0.4 | 1.8 | N/A | 976.9 |
| IP | IIS | 909.0 | 247.4 | 13.1 | 48.2 |
| IP | No-Good | 0.1 | 181.7 | 144.4 | 779.3 |

## G.3 Equal

**Table 14** Summary of Equal Unfiltered Results

| Model | Cut | #Problems | #Optimal | TotalTime | Gap |
|---|---|---|---|---|---|
| DD-JS | IIS | 135 | 84 | 709.9 | 0.4 |
| DD-JS | No-Good | 135 | 72 | 758.8 | 0.6 |
| DD-LJ | IIS | 135 | 67 | 825.2 | 0.6 |
| DD-LJ | No-Good | 135 | 63 | 844.6 | 0.7 |
| IP | IIS | 135 | 17 | 1121.1 | inf |
| IP | No-Good | 135 | 49 | 957.6 | 1.1 |

**Table 15** Summary of Equal Filtered Results

| Model | Cut | OptimalTimes | GapWithoutInf | #Inf |
|-------|-----|--------------|---------------|------|
| DD-JS | IIS | 411.9 | 0.4 | 0 |
| DD-JS | No-Good | 372.1 | 0.6 | 0 |
| DD-LJ | IIS | 444.0 | 0.6 | 0 |
| DD-LJ | No-Good | 415.3 | 0.7 | 0 |
| IP | IIS | 222.9 | 2.7 | 58 |
| IP | No-Good | 529.1 | 1.1 | 0 |

**Table 16** Detailed Equal Results

| Model | Cut | #Callback | #Cuts | ResolTime | ResolTimePerCB |
|-------|-----|-----------|-------|-----------|----------------|
| DD-JS | IIS | 76.9 | 14834.9 | 610.3 | 7.9 |
| DD-JS | No-Good | 98.5 | 23944.3 | 648.7 | 6.6 |
| DD-LJ | IIS | 63.7 | 12501.1 | 756.1 | 11.9 |
| DD-LJ | No-Good | 88.1 | 21152.5 | 737.5 | 8.4 |
| IP | IIS | 7.6 | 149.5 | 99.3 | 13.0 |
| IP | No-Good | 43.1 | 5675.6 | 654.4 | 15.2 |

| Model | Cut | CreateCutTime | CreateSPTime | UpdateTime | ResolTime |
|-------|-----|---------------|--------------|------------|-----------|
| DD-JS | IIS | 0.4 | 0.3 | N/A | 610.3 |
| DD-JS | No-Good | 0.6 | 0.3 | N/A | 648.7 |
| DD-LJ | IIS | 0.3 | 1.7 | N/A | 756.1 |
| DD-LJ | No-Good | 0.6 | 1.6 | N/A | 737.5 |
| IP | IIS | 719.3 | 251.9 | 26.9 | 99.3 |
| IP | No-Good | 0.2 | 178.9 | 109.1 | 654.4 |