Two-stage stochastic/robust scheduling based on permutable operation groups

Louis Rivière, Christian Artigues, Hélène Fargier

Two-stage stochastic/robust scheduling based on permutable operation groups

Louis Riviere*(lriviere@laas.fr)[1,2,3], Christian Artigues[2,3] and Hélène Fargier[1,3]

[1]IRIT, Université de Toulouse, CNRS, UPS, Toulouse, France
[2]LAAS-CNRS, Université de Toulouse, CNRS, UPS, Toulouse, France
[3]Artificial and Natural Intelligence Toulouse Institute, Université de Toulouse, France

Keywords: Stochastic and robust 2-stage scheduling, permutable operation groups, Constraint programming.

Abstract

In this paper we study the performance of a two-stage approach to scheduling under uncertainty making use of sequences of groups of permutable operations. Given a sample set of uncertainty realization scenarios, the goal is to compute a sequence of groups of permutable operations representing a partial scheduling decision in the first-stage, that yields the best possible score in the second-stage, when, for a specific scenario, a full operation sequence is obtained via a second-stage decision policy. This approach is described for a single machine problem and the jobshop problem with stochastic and robust optimization, as well as several commonly studied objectives. We propose new constraint programming models as well as a genetic algorithm meta-heuristic to compute such two-stage solutions. We also investigate a warm-start scheme to work around the difficult search space of sequences of permutable operations. Experiments are carried out to characterize when this two-stage approach yields better results. We also compare the introduced methods with existing ones. Theoretical extensions of the known methods are also described and evaluated.

Statements and Declarations

Funding

Our work was supported by the AI Interdisciplinary Institute ”Artificial and Natural Intelligence Toulouse Institute” (ANITI). ANITI is funded by the French ”Investing for the Future – PIA3” program under the Grant agreement n°ANR-19-PI3A-0004.

Availability of data and materials

Algorithms, instances and raw results presented in this paper are available at :
[https://gitlab.laas.fr/roc/louis-riviere/two-stage-scheduling-using-pogs]
or upon request to the corresponding author.

Other competing interests

The authors have no other relevant financial or proprietary interests in any material discussed in this article.

1 Introduction

Scheduling problems remain relevant today due to their numerous real life applications and their difficult nature. Furthermore Goldratt (1997) showed that the disregard for uncertainty of deterministic scheduling can lead to poor performances in practice. As a result, an abundance of works can be found tackling the problem of scheduling under uncertainty (Aytug, Lawley, McKay, Mohan, & Uzsoy, 2005; Daniels & Kouvelis, 1995; Davari & Demeulemeester, 2019; Herroelen & Leus, 2005; Li & Ierapetritou, 2008; Möhring, Radermacher, & Weiss, 1984; M. Pinedo & Schrage, 1982). Among all these approaches, we are interested in the two-stage setting where proactive decisions are taken off-line when uncertainty prevails while reactive decisions are taken on-line in a second stage when uncertainty is revealed. Two-stage optimization is challenging especially when the deterministic counterpart of the scheduling problem is NP-hard. This paper presents a novel two-stage stochastic/robust optimization approach for scheduling problems under discrete uncertainty scenarios.

In term of scheduling context, we consider two different settings to illustrate the relative generality of this approach : the single machine scheduling problem (1-P) and the job-shop problem (JS-P). In the 1-P, a set of jobs has to be scheduled on a single machine such that each job is characterized by a processing time and a release date. A schedule assigns a start time to each job such that no two job overlap, and each job starts after its release date. A schedule can be evaluated by two different criteria to be minimized, either the the maximum lateness criterion (i.e the maximum for all jobs of the difference between the completion time of the job and its due date), or the sum of completion times criterion.

The JS-P considers a set of jobs and a set of machines. Each job is defined as a chain of tasks, each of them being defined by its processing time and its assigned machine. A schedule assigns a start time of each task such that each task is scheduled after its predecessor in its job chain and the tasks do not overlap on the machines. We also consider two scheduling criteria: the makespan and the sum of completion times.

The deterministic variants of the 1-P and JS-P (for the considered criteria) have been widely studied and all four problems are know to be strongly NP-hard (M. L. Pinedo, 2012). In this paper, we aim at solving uncertain variants of these problems. For the 1-P, the release dates are assumed to be known only in the form of discrete scenarios.

A scenario provides a release date for each job. Such an uncertainty may come for supplier lead-time uncertainty as a typical example (C. W. Wu, Brown, & Beck, 2005). For the JS-P, we consider discrete scenario sets for a more classical processing time uncertainty, where a scenario provides a processing time for each task of each job. We assume that the set of scenario comes from a distribution that is unknown to the decision maker. These scenarios can be sampled from previous executions of the problem (for example previous delivery dates or run-time of tasks). In such an uncertain context, assigning in advance a precise start time to each task is unlikely to lead to an acceptable solution. Indeed, as either the release dates or the processing times are unknown, ensuring schedule feasibility would require extremely conservative estimates, resulting in detrimental costs.

Instead of precise start times, our solutions of uncertain scheduling problems takes the form of a scheduling policy (Möhring et al., 1984) that prescribes during the execution of the schedule and scenario realization which task to execute at each decision time, i.e. times where a machine is free and some tasks are ready for scheduling.

Some scheduling policies are purely dynamic, and make the decision only using the information on the schedule execution and scenario realization. For the 1-P, the earliest release date (ERD) policy (Bachtler, Krumke, & Le, 2020), selects at each decision time, the ready task with the earliest release date. Its equivalent policy for the JS-P is the first-in first-out (FIFO) policy which schedules at each decision time, on an available machine, the tasks that are ready the earliest on this machine. On the opposite side of the spectrum from purely reactive scheduling policies, we find the earliest start (ES) policies (Möhring et al., 1984). They require the addition of precedence constraints such that all resource conflicts are eliminated, which constitutes a static part of the decision, before any information on realized scenario is known. In the 1-P and the JS-P (where resources are unary) this amounts to order all tasks in a sequence on each machine, leaving only the starting dates to be set for each job. We call it the "job sequence" (J-SEQ) policy. Hence each time a machine becomes free, the J-SEQ policy schedules the next task in the sequence at the earliest possible start time.

We can now associate to a scheduling policy a cost equal to the average value of the objective function on the set of scenarios in the context of stochastic scheduling, or the maximum value of the objective function on the set of scenarios in the context of robust scheduling. The problem of finding the optimal J-SEQ policy according to this cost can be defined as a two-stage stochastic/robust optimization problem, where the first stage (here-and-now) variables define the task sequences while the recourse (wait-and-see) variables define the starting times. This problem has been considered in a number of papers in the literature (Akker, Blokland, & Hoogeveen, 2013; Ghasemi, Ashoori, & Heavey, 2021; Gu, Gu, & Gu, 2009; Hao, Lin, Gen, & Ohno, 2013; Horng & Lin, 2015; Kouvelis & Yu, 1997; Wang, Wang, Lan, & Pan, 2018; Wang, Wang, & Xie, 2019).

It can be noticed that the ERD/FIFO policy is purely dynamic in the sense there is no anticipation that would preconstrain the output schedule. On the positive side, it fully adapts to the realized scenario. Its drawback is that it can lead to solutions with high costs. Applied to a deterministic problem, it is equivalent to a simple priority-rule heuristic which is likely to give poor solutions on the considered NP-hard scheduling problems. On the contrary, the JSEQ policy can be considered as mainly static as the off-line computed sequence is strictly followed for any realized scenario. An advantage

is that the sequence can be optimized according to the above-defined stochastic or robust cost. However there is little adaptation to the realized scenario and the cost of the schedule on some individual scenarios can be high even if the average or maximum cost has been minimized.

In this paper, we consider the group-of-permutable-task (G-SEQ) policy that requires, on each machine, the definition of a partial order of tasks in the form of an ordered partition of the tasks, or sequence of groups of tasks. The G-SEQ policy uses the ERD/FIFO policy within each groups, but always schedules the tasks in the first group before scheduling the tasks in the second group etc. The G-SEQ policy dominates both the ERD/FIFO policy and the J-SEQ policy as defining only one group on each machine gives the FIFO policy and defining one group per task gives the J-SEQ policy. Moreover, a compromise between off-line and online decisions can be obtained. We now consider the two-stage robust/stochastic scheduling problem that aims at finding the optimal G-SEQ policy. Even though the solution dominates both J-SEQ and ERD/FIFO policies, it remains to prove that under a limited CPU time a solution method for the G-SEQ two-stage problem is able to find better solutions than the ERD/FIFO policy and than the solution to the J-SEQ two-stage problem. We are also interested in answering the question of how a G-SEQ/J-SEQ solution optimized for a scenario set issued from a distribution adapts to a larger scenario set issued from the same distribution.

Related works are presented in Section 2. The studied problems (1-P, JS-P) are presented in Section 3. Then the J-SEQ and G-SEQ solution schemes are detailed in section 4. Section 5 gives the constraint programming models and the warm-start heuristics for the G-SEQ scheme. Section 6 describes the methods used for the J-SEQ scheme and that both serve as reference methods for comparison with the G-SEQ methods and generate initial solutions for the G-SEQ warm start methods. The computational experiments carried-out are described in Section 7, while the computational results and method comparison are given in Section 8. Concluding remarks are drawn in Section 9.

2 Related work on proactive-reactive scheduling with flexible solution approaches

Among ways to deal with uncertainty, the most complete approaches are certainly ”proactive-reactive” approaches. Such methods try to generate a solution in the proactive ”offline” phase, that allows a good reaction to perturbances in the reactive ”online” phase. This results in an overall better schedule. We call these two phases the first-stage and second-stage decisions.
One such proactive-reactive approach, introduced in (Erschler & Roubellat, 1989), makes use of the sequences of permutable operation groups (G-SEQ). The idea is to produce, in the first-stage, a compact representation of a set of schedules by deciding on a partial order of the tasks to schedule, then allow the second-stage to settle the remaining decisions in order to obtain the final schedule. This second-stage decision can be informed by additional data obtained after first-stage decisions were made.

Although this approach was integrated in an industrial software named ORDO (Billaut & Roubellat, 1996), few works investigate the empirical gain obtained by using

group solutions for problems with uncertainty in lieu of sequences. In (Cardin, Mebarki, & Pinot, 2013; Pinot, Cardin, & Mebarki, 2007), the authors investigate the impact of uncertainty on C_{max} for group sequences when compared to purely predictive or purely proactive approaches. They first compute an optimal schedule for the deterministic problem, then they use the greedy heuristic defined by Esswein (2003) to generate group sequences, aiming to maximise flexibility while guaranteeing an upper bound on solution scores. For the second-stage decisions, they order groups using the FIFO heuristic, starting available tasks first. The permutable operation group models can be defined as a more flexible alternative to sequences. Indeed, only a partial order of jobs is fixed on each machine at first-stage, in the form of a sequence of permutable operations groups (G-SEQ), while both the full job sequences and start times are set depending on the realized scenario at the second-stage.

The approach by S. D. Wu, Byeon, and Storer (1999) can be seen as a pioneering work following these scheme. The authors use group scheduling for minimization of weighted tardiness in a jobshop problem. They use a Branch-and-bound approach to generate G-SEQ, but limit themselves to 2 groups, they schedule the jobs using the ATC heuristic proposed by Vepsalainen and Morton (1987) in the second-stage. In these previous works investigating sequences of permutable jobs for scheduling under uncertainty (Cardin, Mebarki, & Pinot, 2013; Pinot, Cardin, & Mebarki, 2007; S. D. Wu, Byeon, & Storer, 1999), the authors aim in the proactive phase for the most flexible or best bounded solution, the goal being to provide a decision-maker with different options for the reactive phase. In experiments they use a decision heuristic such as FIFO ordering or ATC mentioned earlier.

Artigues, Jean-Charles, Cheref, Mebarki, and Yahouni (2016); Cheref, Artigues, and Billaut (2016a, 2016b) introduce for the single-machine problem a MILP approach to the computation of optimal group sequences w.r.t. the worst-case maximum lateness for a discrete set of scenarios given that the second-stage schedules the jobs according to the earliest release date first policy. A greedy algorithm and a tabu heuristic are proposed for both the J-SEQ and G-SEQ model. The authors provide, for the single machine scheduling problem and on the robust setting only, a partial yet positive answer to the question whether given a limited given computational time, the G-SEQ approach can indeed provide more robust solutions than the standard J-SEQ approach despite the higher problem hardness. The MILP and Tabu approaches searched for the best task or group sequences in terms of worst-case maximum lateness according to a training scenario set. Then J-SEQ and G-SEQ sequences were compared in terms of worst-case maximum lateness on a larger scenario set.

One interest of this paper is to quantify what can be gained if the use of the FIFO decision heuristic is anticipated during the first-stage decision process. More precisely, the goal is not to obtain the most flexible sequence of permutable groups, but instead to reach the solution that will best behave when combined with the given second-stage heuristic (namely FIFO dispatching).

As stated in the introduction, these previous work proposed a framework and presented first results comparing J-SEQ and G-SEQ approaches. There were however several limitations. The MILP proposed for the G-SEQ model in (Artigues et al., 2016; Cheref et al., 2016a, 2016b) is only able to solve very small problems and a single tabu search heuristic was proposed for the G-SEQ approach. Only the robust setting in the 1-P

problem was studied. No comparison with flexibility maximization approaches was carried out. In (Cardin, Mebarki, & Pinot, 2013; Pinot, Cardin, & Mebarki, 2007), only the JS-P is studied, and the first stage decision does not account for the second stage decision. It rather guides the search by maximising solution flexibility, or minimizing the worst-case score.

Building on previous works, the present paper aims at consolidating the comparison between J-SEQ and G-SEQ approaches with several new contributions:

- A new constraint programming model is introduced the G-SEQ schemes.
- A new tabu search and a genetic algorithm are proposed for the G-SEQ scheme, that can be used in a warm-start setting to improve a initial J-SEQ solution.
- The approach is extended to more complex scheduling problems (job-shop), to other objective functions (sum of completion times and C_{max}), and to different uncertainty context (stochastic vs robust)
- An alternative definition of a group sequence is proposed, which does not require each compatible sequence to be valid.
- A comparison with previously proposed flexibility maximization methods is carried out.
- Extensive computational experiments are carried out to compare the new J-SEQ and G-SEQ approaches in a stochastic and robust setting.

List of Notations

Acronyms/stand-ins	**Description**
$1-P$	Single machine problem
CC	Connected component
CP	Constraint programming
CPO	IBM's CP Optimizer
ERD	Earliest release date
EW	Greedy heuristic
$FIFO$	First in first out
GA	Genetic algorithm
$G-SEQ$	Sequence of permutable groups scheme/solution
$INIT$	Time allocated to J-SEQ solver in warm-start heuristics.
$J-SEQ$	Sequence of tasks scheme/solution
$JS-P$	Job shop problem
$MILP$	Mixed integer linear programming
TAB	Tabu method
TPI	Tolerating partially invalid solution parameter
WC	Worst case optimization parameter
WCG	Worst case graph
$WCG*$	Worst case graph with no intra-group arcs

Symbol	Description
$d_i \in D$	Due dates
E	Precedence constraints
G	Precedence graph
$G_i^{(m)}$	The ith permutable group of a G-SEQ
M	Machines
N	Jobs
$p_i^{(s)} \in P$	Process times
$r_i^{(s)} \in R$	Release dates
$s \in S$	Scenarios
v	Velocity
Δ	Variability parameter
γ	Objective type
$\pi^{(m)}$	a G-SEQ
Π	Density of the precedence graph
σ	Objective aggregator among scenarios
$\prec$	Precedes

3 The 2-stage robust/stochastic scheduling problems

In this paper, we are interested in scheduling problems under uncertainty where the uncertainty is modeled as a set S of discrete scenarios. In the following, we will focus on two general problems: a single machine problem with precedence constraints, and a jobshop problem. For each of those problems, we study both a robust optimization scheme and a stochastic one, with several objectives.

3.1 Single machine problem

The single machine problem consists in scheduling a set of jobs N given **uncertain release dates** described by a set of discrete scenarios S. We define a problem with $|N|$ jobs and $|S|$ scenarios as a tuple **1-P** $= (P, R, D, E, \sigma, \gamma)$ where

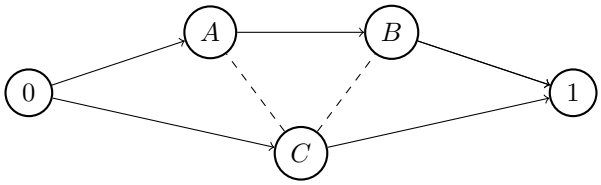
- $p_i \in P$ is the duration of job i.
- $r_i^s \in R$ is the release date of job i in scenario s.
- $d_i \in D$ is the due date of job i.
- E is the set of precedence constraints ($(i, j) \in E$ iff job i must be scheduled before job j).
- Parameter $\sigma \in \{max, avg\}$ is the "objective aggregator" amongst scenarios. Classically $\sigma = max$ corresponds to robust optimization while $\sigma = avg$ corresponds to stochastic optimization.

Job	r_j	p_j	d_j
A	0	2	3
B	3	3	5
C	2	3	5

(a) Scenario 1

Job	r_j	p_j	d_j
A	0	2	3
B	2	3	5
C	3	3	5

(b) Scenario 2



(c) Precedence graph

Fig. 1 Example single machine problem with 2 scenarios: P1

- $\gamma \in \{\sum C_i, L_{max}\}$ is the objective type. Objective $\sum C_i$ corresponds to the sum of completion dates of tasks while L_{max} corresponding to maximum task lateness $(max(0, \max_{i \in N} C_i - d_i))$.

In Figure 1, an problem example P1 with three jobs and two scenarios is given. In the precedence graph, full edges represent precedence constraints, while dotted lines represent undecided order relation between tasks. Examples of first-stage and second-stage decisions in a solution of P1 are given in the J-SEQ setting in Fig. 4 and in the G-SEQ setting in Fig. 6.

3.2 Jobshop problem

The jobshop problem consists in scheduling a set of jobs N on several machines M given uncertain task duration described by a set of discrete scenarios S. Each job is composed of $|M|$ tasks that must run on the machines. Only tasks of a job are subject to precedence constraints. Jobshop problems are defined as a tuple **JS-P** $= (P, M, \sigma, \gamma)$ where

- $p^s_{i,j} \in P$ is the duration of task j of job i in scenario s.
- $m_{i,j} \in M$ is the machine on which task j of job i must run.
- $\sigma \in \{max, avg\}$ is the same scenario aggregator.
- $\gamma \in \{\max_{i \in N} C_i, \sum C_i\}$ is the objective type.

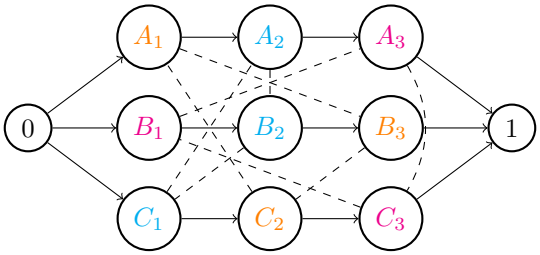
Figure 2 gives an example with three jobs, three machines and two scenarios. Machines are depicted with colors and dotted lines.

p_i^j	1	2	3
A	3	2	1
B	2	2	1
C	1	2	1

(a) Scenario 1

p_i^j	1	2	3
A	2	2	1
B	3	2	1
C	1	2	1

(b) Scenario 2



(c) Precedence graph

Fig. 2 Example jobshop problem with 2 scenarios: P2

3.3 Generic 2-stage problem statement

In both cases, we consider a 2-stage robust or stochastic scheduling problem where $\mathcal{X}$ is the set of first-stage decisions while $\mathcal{Y}(x, s)$ is the set of second-stage decisions given a scenario s that actually occurs and a first-stage decision x. Then the problem can be stated as follows:

$$\min_{x \in \mathcal{X}} \sigma_{s \in S} \min_{y \in \mathcal{Y}(x,s)} \gamma(y, s)$$

where $\sigma \in \{avg, max\}$ and $\gamma \in \{L_{max}, \sum C_i\}$ for the 1-machine problem and $\gamma \in \{C_{max}, \sum C_i\}$ for the job-shop problem. This gives us 8 different scheduling problems with discrete uncertainty scenarios.

The definition of the first-stage decisions $\mathcal{X}$ and second-stage decisions $\mathcal{Y}$ are slightly different in the J-SEQ and G-SEQ models. They will be defined in the following sections.

4 First-stage solution representations and second-stage policy

Solutions to scheduling problems are usually schedules, i.e a start time for each job. But with uncertain parameters, such solutions might either be invalid or too conservative. That is why we study 2-stage solution methods. In the first-stage, a flexible solution is found, then when a scenario occurs, the second-stage policy adjusts the schedule.

We consider two different settings for the first-stage decision $\mathcal{X}$: respectively job sequences (J-SEQ) and sequences of groups of permutable jobs (G-SEQ). For both first-stage solution representations, we define a valid sequence. The definition is rather

straightforward for J-SEQ but can have multiple meanings for G-SEQ.
The valid first-stage J-SEQ and G-SEQ are defined in sections 4.1 and 4.2, respectively. The second-stage decision policy, i.e. the way a first-stage decision is extended to a second-stage decision via a scheduling policy $\mathcal{Y}$ upon progressive scenario realization in real-time is presented in section 4.3.

4.1 Valid job sequences

A job sequence (J-SEQ) is a total ordering of the jobs on each machine, i.e a $|M|$-vector of job sequences. This allows to use the same terminology for the single-machine problem and the job-shop problem. The set of first-stage decisions $\mathcal{X}$ is the set of job sequences. A J-SEQ is valid if its associated left-shifted schedule satisfies all constraints. To define formally a valid J-SEQ both in the single-machine and job-shop setting, the disjunctive graph is a convenient representation (Balas, 1969; Roy & Sussmann, 1964). The disjunctive graph contains a node for each job, plus dummy source and sink nodes. It is composed of directed and undirected arcs. It can be used to represent both the single machine and job-shop scheduling problem as follows:

- The source node is connected to each task node by a directed arc valuated by the task release date for the single-machine case, and by 0 for the job shop.
- Each task node is connected to the sink node by a directed arc valuated by $p_i - d_i$ for the single machine case and by p_i for the job shop.
- For each precedence constraint (i, j) there is an arc directed from node i to node j, valuated by p_i.
- For each pair of task sharing the same machine there is an undirected arc linking the two task node.

Such undirected arcs are also called disjunctive arcs as each of them represents the disjunction between two possible precedence constraints linked to machine usage.
A partial selection is an orientation of a part of the disjunctive arcs. Each directed arc issued from a disjunctive arc is valuated by the duration of the origin task node. A complete selection is an orientation of all disjunctive arcs. A selection is acyclic if it does not induce a cycle in the graph made of the original directed arcs plus the directed arcs issued from the selection. A J-SEQ defines a unique complete selection by orientating each disjunctive arc as prescribed by the job sequence. The following definition of the validity of a J-SEQ is well-known:

Definition 1. *A J-SEQ is valid iff its associated complete selection is acyclic.*

Note that on the single machine problem, this simply amounts to verify that for any pair $(i, j) \in E$, $i \prec j$ in the sequence.

Given a J-SEQ and a scenario, the corresponding left-shifted schedule is among the best possible schedule for regular objectives (Kouvelis & Yu, 1997). It is easily

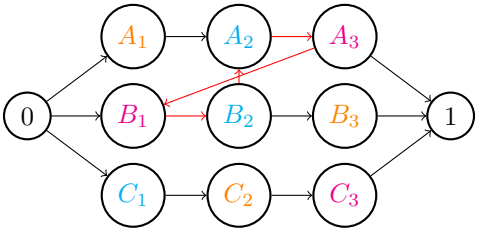
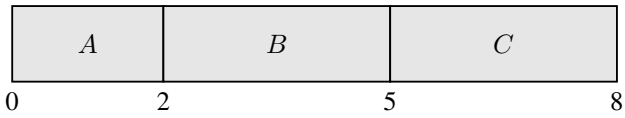
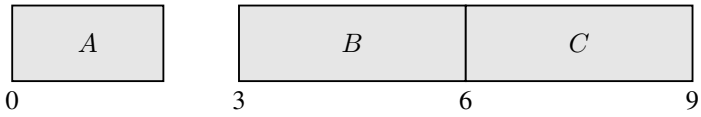


Fig. 3 Example of an invalid J-SEQ



(a) Realisation under scenario 1



(b) Realisation under scenario 2

Fig. 4 The 2 stage J-SEQ solution $A|B|C$ on problem P1 (Figure 1)

obtained since the earliest start time of a task compatible with the J-SEQ is equal to the longest path length between the source node and the task node in the graph issued from the associated complete selection (Balas, 1969), with the particular case of the longest path length between the source and the sink node being equal to the C_{max} for the job-shop problem and the maximum lateness for the one-machine problem. Therefore, the second-stage decision simply follows the J-SEQ and starts tasks as soon as they are ready in that scenario.

Figure 4 displays a solution for the problem P1. In the first scenario, task C is the latest, yielding a maximum lateness of 3, while in the second scenario, task C is also the latest with a maximum lateness of 4. It can be verified that sequence $A|B|C$ is the optimal first-stage solution in the J-SEQ framework as it minimizes the maximum lateness over the the two scenarios, which yield second stage schedules (a) and (b) for scenarios 1 and 2, respectively.

4.2 Valid group sequences

4.2.1 Definitions

A sequence of groups of permutable jobs (G-SEQ) is constituted on each machine $m \in M$ of an ordered partition of the set of jobs to be scheduled on this machine $\pi^m = G_1^m|G_2^m|\ldots|G_q^m$ with $G_q^m \subseteq N$ for all $m \in M$ and $q = 1, \ldots, k^m$; $G_q^m \cap G_{q'}^m = \emptyset$ for all $m \in M$, $q = 1, \ldots, k^m - 1$ and $q' = 1, \ldots, k^m$, $q \neq q'$; $\bigcup_{q=1..k^m} G_q^m = N$, where

k^m denotes the number of groups on machine $m \in M$ and G_q^m with $q = 1, \ldots, k^m$ denotes the qth group on machine m. In this model, the set of first-stage decisions $\mathcal{X}$ is the set of group sequences. It can also be understood as a set of partial orders of the tasks represented by groups on each machine. A G-SEQ represents a potentially large set of job sequences.

In previous approaches based on G-SEQ (Artigues, Billaut, & Esswein, 2005; Artigues, Jean-Charles, Cheref, Mebarki, & Yahouni, 2016; Billaut & Roubellat, 1996; Cheref, Artigues, & Billaut, 2016a, 2016b; Erschler & Roubellat, 1989), a G-SEQ is a valid solution iff **any** J-SEQ constructed by ordering the tasks within a group is valid in the sense of Definition 1. For the single machine problem, that means that a G-SEQ is valid iff for any pair $(i, j) \in E$, the group of job i is ordered strictly before the group of job j in the G-SEQ. However we extend the usual validity definition to distinguish more special cases:

Definition 2. *A G-SEQ is*

- *fully valid if it represents only valid J-SEQ,*
- *fully invalid if it represents no valid J-SEQ,*
- *partially valid if it represents at least one valid J-SEQ,*
- *partially invalid if it represents at least one invalid J-SEQ,*

where a J-SEQ is valid or invalid in the sense of definition 1.

Note that when the G-SEQ is fully valid, it represent $\Pi_{m \in M} \Pi_{q=1..k^m} |G_q|!$ different valid J-SEQs (i.e. complete selections).

4.2.2 Necessary and sufficient conditions

In Artigues, Billaut, and Esswein (2005), a so-called worst-case graph (WCG) is defined for a given G-SEQ and a scenario. The graph contains a source node, a sink node and two nodes i and i' for each task i, representing the start and the completion of the task, respectively. For each task i, there is an arc between the source node and i valuated by r_i, an arc from i to i' valuated by p_i and an arc from i' to the sink node, valuated by $p_i - d_i$. For each precedence constraint between i and j there is an arc from i' to j valuated by 0. For any ordered pair of tasks i, j that belong to two consecutive groups, there is an arc from i' to j valuated by 0. Finally, for any pair of distinct tasks i and j belonging to the same group G, there is an arc from i to j' valuated by $\sum_{k \in G} p_k$. This worst-case graph can be used to compute worst-case completion times for a given G-SEQ and a given scenario as it will be recalled in section 4.2.3. However, from (Artigues, Billaut, & Esswein, 2005) its non valued variant can be used to establish in polynomial time the full validity of a G-SEQ independently of the scenario set, as stated below.

Theorem 1. *(from (Artigues, Billaut, & Esswein, 2005)) A G-SEQ is fully valid iff its non-valued worst-case graph is acyclic.*

Cycle check can be performed by a simple depth first search in the worst-case graph. Note that, by contraposition, the worst-case graph has a cycle iff the G-SEQ is partially invalid.

In the next section we present the second-stage policy that exploit either a fully valid or a partially valid G-SEQ to obtain a schedule compatible with the realized scenario. This will also give us a polynomial algorithm to check whether a given G-SEQ is partially valid.

Theorem 2. *A G-SEQ is fully invalid iff its worst-case graph, with intra-group arcs removed (WCG*) is cyclic.*

$\Longleftarrow$ **:**

Proof. Removing intra-group arcs in the WCG associated with a G-SEQ π leaves only the machine-wise precedence decided by the G-SEQ (the partial selectino) π, the job-wise precedences, and the arcs between nodes i and i'. If there is a cycle in this graph, there is a cycle in the associated disjunctive graph without disjunctive arcs. Hence, no selection of the disjunctive arcs can be acyclic. Therefore π is fully invalid. □

$\Longrightarrow$ **:**

Proof. Let the G-SEQ π be fully invalid. Assume the associated WCG* is acyclic. We will show a contradiction. As stated before, if the WCG* is acyclic, the disjunctive graph (without disjunctive arcs) is acyclic. Furthermore, it is always possible to orient an arc in a graph without introducing a first cycle. Indeed, when orienting arc (i, j), if there is a path from i to j, orienting the arc from i to j doesn't add a cycle unless there is a path from j to i (which would mean the graph already had a cycle, which it doesn't). Otherwise, orienting the arc from j to i doesn't introduce a cycle. Finally, we saw that an acyclic orientation of the disjunctive arcs corresponds to a valid J-SEQ. So π would be partially valid, a contradiction. □

By contraposition, the modified worst-case graph is acyclic iff the G-SEQ is partially valid.

4.2.3 Worst-case score of valid G-SEQ

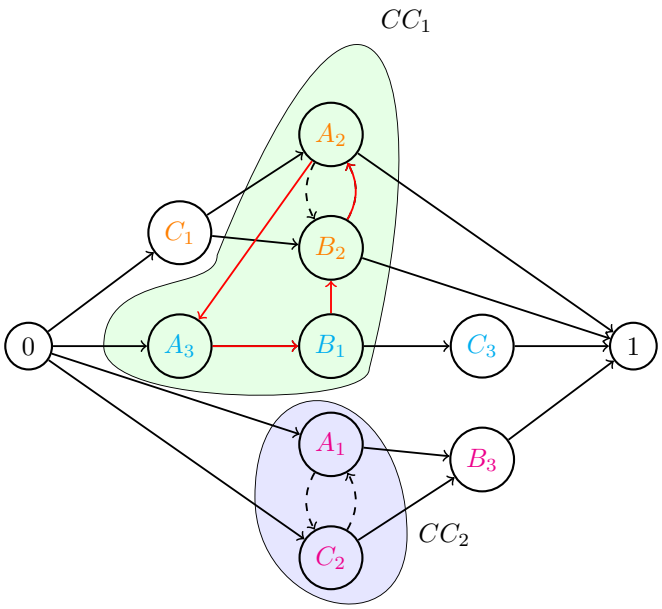
One of the interesting properties of G-SEQ solutions is the ability to find, for each task, its worst case starting time in a given scenario. Indeed, Artigues, Billaut, and Esswein (2005) show that it is possible, using a polynomial algorithm, to find out the largest starting time of a task that can be obtained by left-shifting the task in a J-SEQ represented by the G-SEQ on this scenario. In turn, this allows to find the worst case objectives for max-type regular objectives (such as the maximum lateness $L_{\max}$ or the makespan $C_{\max}$) or an upper bound of the worst case sum-type objective (such as the sum of completion times $\sum C_i$ or the total tardiness $\sum T_i$).
This algorithm is based on finding longest paths in the worst-case graph: the longest path from the source node to node i' is the largest end time of task i in any semi-active

schedule represented by the G-SEQ. This algorithm no longer functions in the case of partially invalid G-SEQs that can be encountered in the jobshop problem (as cycles get introduced in the worst-case graph). To address this issue, we next describe an upper bound algorithm for the worst case starting time of jobs.

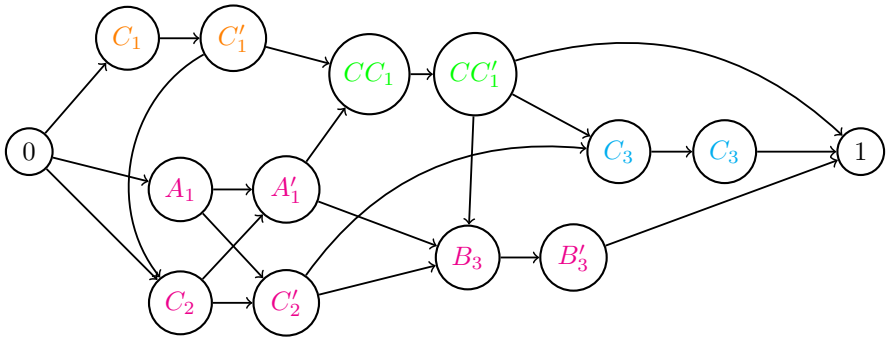
Algorithm 1 Generalized Worst-case algorithm

1: Get the precedence graph G associated with π
2: **if** G has a cycle **then**
3: **return** $\perp$ $\{\pi$ is fully invalid$\}$
4: **end if**
5: Add the bidirectional arcs corresponding to the groups of π to G
6: Get the connected components C of G
7: Modify C by splitting each connected component $c \in CC$ containing only tasks of the same group in $|c|$ elements
8: Build a worst-case graph W using π and C.
9: **return** the worst-case start times in W as in (Artigues, Billaut, & Esswein, 2005)

Algorithm 1 starts with the construction of the precedence graph G associated with the G-SEQ π. G contains only the precedence constraints that are necessarily present in π, hence the tasks within a group are not linked by any arc. G contains a cycle iff π is fully invalid (Theorem 2). Otherwise, the algorithm can move forward. Bidirectional arcs are added to G between all tasks of the same group, yielding the corresponding disjunctive graph. Tarjan's algorithm is then used to get the list C of the connected components in G. An example of this step is given in figure 5a which represents the disjunctive graph associated with the G-SEQ solution $\pi = [\pi^0 = C_1|B_2, A_2, \pi^1 = A_3|B_1|C_3, \pi^2 = A_1, C_2|B_3]$ for some jobshop problem with 3 machines and 3 jobs (some job-wise precedence arcs were omitted for clarity). The connected component analysis yields two components: CC_1 and CC_2.
List C is then modified to only keep the connected components that contain tasks belonging to at least two different machines (in our example, only CC_1 remains). If, after this split, list C contains only single-task components, the G-SEQ is fully valid (In our case, we have a partially invalid G-SEQ). Finally, the WCG is contructed using the components in C (see Figure 5) by considering all tasks in a component c as a single task that inherits all the precedence constraints of the tasks in c, which by construction shall not yield any cycle. Worst case end times for tasks within a cluster are all set to the worst case end time of the cluster, and the process time of the cluster of tasks is set to the sum of the durations of the tasks contained therein. It is interesting to note that some tightness can be lost when grouping tasks together, but none is lost if the solution does not require grouping (i.e. the solution is fully valid). The complexity of this algorithm remains in $O(n^2)$.
Further generalisation is required to compute the worst case starting times for our multi-scenario problem. Hence, the above described worst case algorithm is applied for every scenario, yielding worst case objective in every scenario, that can then be evaluated using aggregator σ, with overall complexity $O(S.n^2)$.



(a) Disjunctive graph and connected component (simplified) for $\pi = [[C_1|B_2, A_2], [A_3|B_1|C_3], [A_1, C_2|B_3]]$



(b) Aggregated worst-case graph

Fig. 5 Illustration for Algorithm 1

Whether finding the worst case starting time of a task is NP-Hard when the G-SEQ can be partially invalid is an interesting open problem.

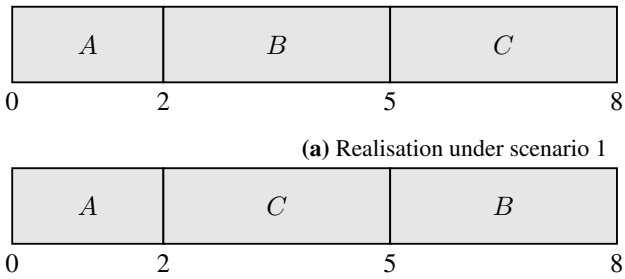
4.3 Second-stage policy

Scheduling is a multi-stage process where uncertainty is generally revealed progressively over time. In this paper, two types of uncertain parameters are considered: the task release dates for the single-machine problem and the task processing times for the job-shop problem. We assume that the release dates are never revealed in advance, but only revealed as soon as the task is released (e.g in scenario s, release date of task i is revealed at time $t = r_i^s$, the release date of that task in scenario). Similarly, the processing time is revealed only at the completion time of each task. It follows that the task start times (i.e. the second-stage variables) can only be assigned via an on-line scheduling policy that prescribes what must be done at each completion time and release date event.

For the J-SEQ model, the on-line scheduling policy simply follows the prescribed sequences, as in a majority of previous works on stochastic or robust job-shop scheduling problems (Akker, Blokland, & Hoogeveen, 2013; Ghasemi, Ashoori, & Heavey, 2021; Gu, Gu, & Gu, 2009; Hao, Lin, Gen, & Ohno, 2013; Horng & Lin, 2015; Kouvelis & Yu, 1997; Wang, Wang, Lan, & Pan, 2018; Wang, Wang, & Xie, 2019). At each time t where a task is completed on a machine, the next planned task in the sequence is scheduled immediately if it is already available, or the machine waits for its earliest availability (i.e. its realized release date for the single machine problem or the realized completion time of its job predecessor for the job-shop problem). For J-SEQ, the second-stage decision set $\mathcal{Y}(x, s)$ is the set of operation start time compatible with the job sequence x and scenario s. Each start time assigned by this policy is equal to the length of the longest path between the source node and the task node in the graph issued from the complete selection induced by the J-SEQ.

For the G-SEQ model, the on-line policy is slightly more complex and is called FIFO policy (Algorithm 2). The second-stage decision set $\mathcal{Y}(x, s)$ is the set of job sequences and operation start times compatible with the group sequence x, obtained as follows. At each time t where a task i is completed, the next selected task is either the earliest available task in the same group as i, or, if all tasks in that group were scheduled already, the earliest available task in the next planned group. The selected task is scheduled as early as possible.

In Figure 6, for the example problem P1, a G-SEQ solution provides as a first stage decision a sequence of two groups, the first one contains only one task, A, and the second one contains B and C. So, for the second stage task A is scheduled in the first position at time 0 for both scenarios. At time 2 task A completes. As there are no more tasks in the group of A, the earliest available task in group $\{B, C\}$ is started next, at its earliest availability time. This differs according to the scenario. In scenario 1, task B is released next, at time 2, which yields the first displayed schedule issued from sequence "ABC". On scenario 2, it is task C that is released next, at time 2, which yields the second displayed schedule issued from sequence "ACB". Compared to the J-SEQ solution of figure 4, this allows to improve the $\max L_{max}$ objective from 4 to 3.



(a) Realisation under scenario 1

(b) Realisation under scenario 2

Fig. 6 The 2 stage G-SEQ solution $\pi = A|B, C$ on problem P1 (Figure 1)

Algorithm 2 FIFO policy

1: **for** $t = 0$ and every following time t when a task finishes **do**
2: **for** Each idle machine $m \in M$ **do**
3: Gather the set of tasks $A = R \cap \mathcal{G}$ that can run on m.
4: Schedule the task of A that was ready first (if any).
5: **end for**
6: **end for**

We notice that J-SEQs constitute a special case of G-SEQs, where each group contains a single job only. As such, G-SEQs in theory dominate J-SEQs, but due to the much larger search space, in a limited computational time, it is not easy to predict which approach will give the best results for practical purposes.

We will show next that iff a G-SEQ contains at least one valid J-SEQ (it is partially valid), the second-stage heuristic will yield a valid J-SEQ. Previously, we have described how a cycle in the precedence graph can lead to an infeasible solution. During the online phase, this manifests by the inability to run any task without violating either the task order within a job, or the task order within planned sequence. Note that this infeasibility happens regardless of the scenario, it is purely dependent on the sequence.
At any given moment during the online phase, let D be the set of tasks that are completed, let R be the set of job-wise ready tasks (i.e they are the first tasks in a job or their predecessors are in D). Let also $\mathcal{G}$ be the set of current groups, i.e, for each machine the earliest group such that some tasks in the group are not in D (planning-wise ready tasks). Let $\mathcal{G}^-$ and $\mathcal{G}^+$ be the set of groups sequenced before and after the groups in $\mathcal{G}$. Let $H : (\text{G-SEQ}, s) \longrightarrow \text{J-SEQ}$ be a second-stage heuristic such that if a task t is both ready to run precedence-wise and planning-wise (i.e. $t \in R \cap \mathcal{G}$), H schedules t in finite time. Note that the FIFO policy defined in Algorithm 2 possesses this property.

Proposition A G-SEQ π is partially valid iff $\forall s \in S$, $H(\pi, s)$ is valid.

Proof. Suppose $s \in S$ is a scenario such that $H(\pi, s)$ is invalid, i.e at one point H

cannot decide of a task to be scheduled (because there is a cycle). It must mean that $R \cap \mathcal{G} = \emptyset$ (no task is ready), else H would eventually launch a task. This implies that there is a task in every current group that has a predecessor in a group not yet ready: $\forall g \in \mathcal{G}, \exists t_i \in g$ such that $\exists t_j \in \mathcal{G}^+, j < i$. The existence of a set $\mathcal{G}$ satisfying this property makes $H(\pi, s)$ invalid $\forall s$. Moreover, because all the tasks in a group share the same precedence w.r.t tasks outside the group on the same machine, no ordering of the groups of $\mathcal{G}$ can lead to a valid sequence, hence there is no valid J-SEQ in π (π is fully invalid). By contraposition, we have that a partially valid G-SEQ always yields a valid J-SEQ through H. □

5 G-SEQ solution approaches

In this section, we describe the methods used to compute G-SEQ and J-SEQ-based solutions to the 8 different scheduling problems with discrete uncertainty scenarios described in section 3.3.

For the sake of conciseness, we mainly describe methods used to solve the single machine problem, and only stress briefly the differences in method for the jobshop problem.

5.1 Constraint programming models

In this section we present a constraint programming (CP) model for each problem. For both the G-SEQ and the J-SEQ models, we let each job have a different start time in each scenario, but all start times must be consistent with a unique J- or G-SEQ across all scenarios. The following CP models (Algorithm 3) describes the key constraints in the CP model used for G-SEQ computation using IBM CP Optimizer (CPO) modeling (respectively for 1-P and JS-P). Expressions and constraints specific to the CPO language are typeset in small capitals.

In model (CP-G-SEQ-1P), variable g_i is an integer denoting the index of the group that contains task i; $Job[i, s]$ is an interval variable embedding the schedule time of task i in scenario s; and u_g is a boolean variable symbolizing that a group index g is used. Lines (2-12) create a CPO model for each scenario. For a given scenario, uncertain data is known, allowing to set usual release date constraints (2). All Tasks (the set of all tasks of a scenario s is referred to as $Job[:, s]$)) are added to a SEQUENCE variable and the NOOVERLAP constraint is applied on the SEQUENCE to enforce task sequencing on the machine (3). Then, every pair of task is considered. If a precedence constraint links them, the predecessor is forced to be in a group of smaller index than the successor (5). If a task i is in a group of smaller index than the group of another task j, implication constraint (6) enforce the precedence constraint $i \prec j$. This synchronizes all scenarios models according to the group sequence represented by variables g. If two tasks are in the same group, the task with the smallest release date in the considered scenario is scheduled first (7–10), as prescribed by the FIFO policy.
Because only the order of groups is a relevant information, and not their indices, we can break a symmetry by forcing the group with the smallest index to be used first. In order

Algorithm 3 CP model for G-SEQ computation in the single machine problem

1: **for** $s \in S$ **do**
2: $\quad$ STARTOF$(Job[i,s]) \geq r_i^s \quad \forall i$
3: $\quad$ NOOVERLAP(SEQUENCE$(Job[:,s]))$
4: $\quad$ **for** $(i,j) \in N^2$ **do**
5: $\quad\quad$ $(i,j) \in E \rightarrow g_i < g_j$ $\{g_i$ is the group number of job $i\}$
6: $\quad\quad$ $g_i < g_j \rightarrow$ ENDBEFORESTART$(Job[i,s], Job[j,s])$
7: $\quad\quad$ **if** $Release(i,s) \leq Release(j,s)$ **then**
8: $\quad\quad\quad$ $g_i == g_j \rightarrow$ ENDBEFORESTART$(Job[i,s], Job[j,s])$
9: $\quad\quad$ **else**
10: $\quad\quad\quad$ $g_i == g_j \rightarrow$ ENDBEFORESTART$(Job[j,s], Job[i,s])$
11: $\quad\quad$ **end if**
12: $\quad$ **end for**
13: **end for**
14: **for** $g \in N$ **do**
15: $\quad$ $u_g = \vee(g_i == g, \forall i \in N)$ $\{$ u_g marks a group as "used"$\}$
16: $\quad$ **if** $g > 0$ **then**
17: $\quad\quad$ $u_{g-1} \geq u_g$
18: $\quad$ **end if**
19: **end for**

to enforce that, lines (14–18) use the boolean variable u_g. Line (15) makes u_g take the value of "1" iff the group is used, that is if any task is in group g. Then constraints (16–18) allow a group to be used (u_g value of "1") only if it is the group 0 or if the previous group is used. As a result, if k groups are used, they must be the k first groups (0 to $k-1$).

The job-shop structure makes the CP model more complex than for the single machine problem. It is described in (Algorithm 4):
A variable g_i^m gives the group index of each job i on each machine m. As previously, a model is defined for each scenario, each model being synchronized via the g variables. Constraints (2) are the precedence constraints between operations of the same jobs are enforced. A SEQUENCE variable is created for each machine and a NOOVERLAP constraint on each SEQUENCE ensures full sequencing of the tasks on the corresponding machine (3). Additional "ready date" integer variables $Ready[m,i,s]$ are needed. They keep track of the time at which a task's predecessor is completed, as enforced by constraints (4-5). For each machine m and each pair of jobs $\{i,j\}$, FIFO sequencing of the operations of these jobs on machine m is enforced depending on group indices g_i^m and g_j^m, as for the single machine case via constraints (6–15). Note that, in contrast to the single machine problem, in the jobshop problem ready dates variables are used instead of the release dates to enforce order of operations within groups..

Constraints (16–18) are needed to enforce operations left-shifting. Otherwise the solver would be allowed to delay running tasks in order to influence ready dates and, in turn, future order decisions of tasks within a group, which goes against the non-anticipation hypothesis in stochastic and robust frameworks, as stated by the following

Algorithm 4 CP model for G-SEQ computation in the jobshop problem

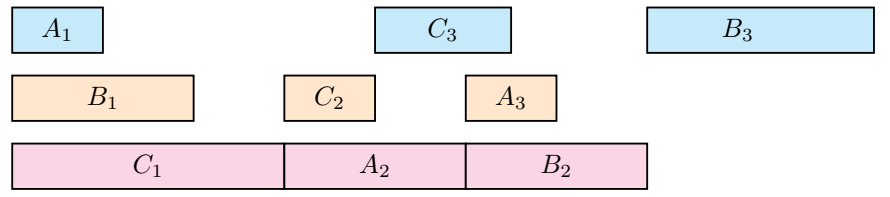
for $s \in S$ **do**
  STARTOF($Job[m_{i,j-1}, i, s]) \geq EndOf(Job[m_{i,j}, i, s]) \quad \forall i \in N, \forall j \in M \setminus \{0\}$
  NOOVERLAP(SEQUENCE($Job[m, :, s])) \forall m \in M$
  $Ready[m_{i,0}, i, s] == 0 \quad \forall i \in N$
  $Ready[m_{i,j}, i, s]$ ==ENDOF($Job[m_{i,j-1}, i, s] \quad \forall i \in N, \forall j \in M \setminus \{0\}$
  for $m \in M$ **do**
    for $(i, j) \in N^2$ **do**
      $g_i^m < g_j^m \rightarrow$ ENDBEFORESTART($Job[m, i, s], Job[m, j, s]$)
      if $Ready(m, i, s) \leq Ready(m, j, s)$ **then**
        $g_i^m == g_j^m \rightarrow$ ENDBEFORESTART($Job[m, i, s], Job[m, j, s]$)
      else
        $g_i^m == g_j^m \rightarrow$ ENDBEFORESTART($Job[m, j, s], Job[m, i, s]$)
      end if
    end for
  end for
  for $m \in M$ **do**
    STARTOF($Jobs[m, i, s]$) ==MAX(ENDOFPREV(SEQUENCE$[m]$, $Jobs[m, i, s]), Ready[m, i, s]), \forall i \in N$
  end for
end for
for $m \in M$ **do**
  for $g^m \in N$ **do**
    $u_g^m = \vee(g_i^m == g^m, \forall i \in N)$
    if $g^m > 0$ **then**
      $u_{g-1}^m \geq u_g^m$
    end if
  end for
end for

theorem:

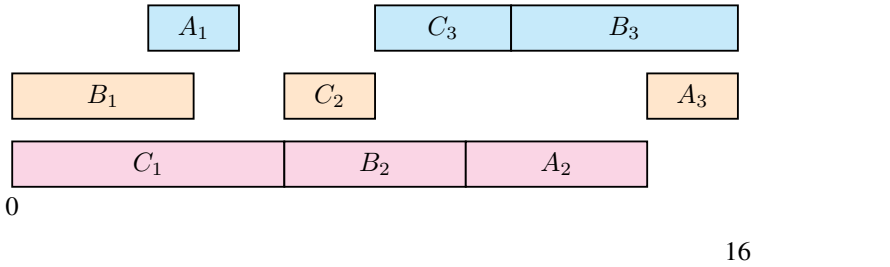
Theorem 3. *Given a G-SEQ solution and the FIFO policy inside each group, semi-active schedules are not dominant for C_{max} minimization.*

Proof. Figure 7 shows off the unwanted behavior that can occur when constraints (16–18) are missing. Figure 7a displays expected behavior when considering solution $\pi = [[A_1|C_3, B_3], [B_1|C_2|A_3], [C_1|A_2, B_2]]$ for a problem with 3 jobs and 3 machines: tasks A_1 and B_1 are started without delay, hence task A_1 finishes first, and when considering the group $G_2^3 = [A_2, B_2]$, the FIFO heuristic schedules A_2 first. However, this turns out to be detrimental for the objective, running task B_2 first allows for a better C_{max}. Knowing this, the solver left unchecked will delay task A_1 as in figure 7b such that it finishes after B_1, and allowing the FIFO heuristic to launch B_2 before A_2 in order to reach the better C_{max}. □

In the policy associated to real time decision making, delaying tasks in order to improve the solution, as described above, requires the knowledge of the whole scenario data. This clearly violates the non-anticipation hypothesis : task processing times are only revealed upon operation completion. The ENDOFPREV expression represents the completion time interval variable that precedes the job given in parameter in the SEQUENCE variable also given in parameter. Hence, constraints (17) set the start time of each operation to the maximum between the completion time of its predecessor on its machine and the completion time of its predecessor in the jobs, which amounts to a "manual" left shift.



(a) Semi-active schedule schedule



(b) Non semi-active schedule

Fig. 7 Non dominance of semi-active schedules for G-SEQ solutions

Note that both models encapsulate the online scheduling policy associated with selected G-SEQ solutions.

5.2 Warm start heuristics

As we will see in section 8, even though G-SEQs are better in theory, results show that in limited time, for larger instances, CPO gets much better results using the J-SEQ model than the G-SEQ model. In order to take advantage of the good performance of the J-SEQ solver, but still retaining some of the possible gain and flexibility of G-SEQs, we then introduce heuristic G-SEQ methods based on a good J-SEQ starting solution. We will compare the J-SEQ solution at timeout with several heuristics, all of which consist in running the J-SEQ solver for part of the time limit, and then using the best found J-SEQ solution so far as an input to a G-SEQ heuristic.
We compare 3 warm start methods:

- The greedy heuristic of Esswein (2003) (EW-G-SEQ).
- The tabu search algorithm of Artigues, Jean-Charles, Cheref, Mebarki, and Yahouni (2016) (TAB-G-SEQ).
- A new genetic algorithm (GA-G-SEQ).

Additionally, we compare those methods when combined with the following parameters:

- Worst-case optimization (WC=True): when optimizing using the worst-case, the methods aim to find the G-SEQ solutions such that the worst J-SEQ it represents has the best score over all training scenarios (the more common approach in literature (Cardin, Mebarki, & Pinot, 2013; Esswein, 2003; Pinot, Cardin, & Mebarki, 2007)). Recall that the worst-case objective of a fully valid group sequence on a given scenario can be computed exactly for the C_{max} and $L_{\max}$ objectives or approximately for the $\sum C_i$ criterion in polynomial time using longest path computations the worst-case graph (Artigues, Billaut, & Esswein, 2005). For partially valid group sequences, the worst case objective is approximated using the algorithm described in section 4.2.3. Optimizing without worst-case (WC=False) instead takes into account the second-stage heuristic and aims to find the G-SEQ solution that will have the best score over all training scenarios, when using the second-stage heuristic (as in (Cheref, Artigues, & Billaut, 2016b)).
- Tolerating partially invalid solutions: For the jobshop problem, we also study the impact of allowing (TPI=true) or not (TPI=false) partially invalid solutions, which are usually not allowed as solutions in the literature.
- Init time (INIT): The time allocated to the initial run of the J-SEQ solver. The remaining time is the running time of the warm start method, starting from the J-SEQ solution.

5.2.1 Tabu search algorithm : TAB-G-SEQ

We used a variant of the tabu search method proposed by Artigues, Jean-Charles, Cheref, Mebarki, and Yahouni (2016); Cheref, Artigues, and Billaut (2016b) in which we start from a J-SEQ solution (instead of a G-SEQ solution computed by a greedy heuristic as in (Artigues, Jean-Charles, Cheref, Mebarki, & Yahouni, 2016; Cheref, Artigues, & Billaut, 2016b)). The method explores at each step 4 neighborhoods:

- Group swap: swaps the content of two groups (all task in a group i now belong to group j and vice versa)
- Group insert: moves a single task from one group to another
- Group split: splits a group G into two groups G^- and G^+. There are $2^{|G|}$ ways to split a group into two, hence for efficiency, we define a split factor for a group and only the $|G|$ splits where tasks in G^- have a smaller average split factor over all scenarios than tasks in G^+ are considered. Because of our problem uncertain data, we compute the split factor as the average release date on the 1-P and the average sum of predecessors running time on JS-P. The idea being that having the tasks likely to be ready first in the first group G^- should introduce less idle time.
- Group fusion: merges two consecutive groups.

A tabu list of length $10 * N$ is used but experiments (Artigues, Jean-Charles, Cheref, Mebarki, & Yahouni, 2016) showed it had little impact on the performance of the heuristic. The procedure stops at the time limit.

5.2.2 Greedy heuristic : EW-G-SEQ

The greedy heuristic EW-G-SEQ, named EBGJ in (Esswein, 2003), starts from a J-SEQ. Then, only the merging of consecutive groups is considered. Groups are merged trying to minimize the score (or not increase it in the case of worst case score) and prioritizing the merging of smaller groups. The procedure stops when no merge can be done without increasing the score, or at timeout (see Algorithm 5).

Algorithm 5 Overview of the greedy heuristic EW-G-SEQ

1: From a starting solution x
2: **while** there are non-worsening neighbors and within time limits **do**
3: Compute the set $\mathcal{N}$ of valid "group merge" neighbors.
4: x becomes the best neighbour in $\mathcal{N}$ (by minimal score, then minimal size of the largest group).
5: **end while**
6: Return x

5.2.3 Genetic algorithm : GA-G-SEQ

The genetic algorithm implemented is described in Algorithm 6.

Algorithm 6 Overview of Genetic algorithm GA-G-SEQ

1: $Population \leftarrow InitPopulation(PopSize)$
2: **while** $(time < MaxTime) \wedge (stagnate < MaxStagnate)$ **do**
3: $A, B \leftarrow Select2(Population)$
4: $C \leftarrow Crossover(A, B)$
5: **if** $Invalid(C) \wedge (random() < RepairRate)$ **then**
6: $C \leftarrow Repair(C)$
7: **end if**
8: **if** $random() < MutateRate$ **then**
9: $C \leftarrow Mutate(C)$
10: **end if**
11: **if** $random() < EducateRate$ **then**
12: $C \leftarrow Educate(C)$
13: **end if**
14: $Population \leftarrow Population + C$
15: **if** $PopulationLimit < Population$ **then**
16: $Population \leftarrow SelectSurvivors(Population)$
17: **end if**
18: **end while**

First, an initial population is created (line 1). It contains the good starting J-SEQ solution, as well as $PopSize - 1$ randomly generated G-SEQ solutions. Second, two individuals are selected at random and crossed using an extension of the usual 1 point crossover for sequences (lines 3-4)(see Fig. 8: π_1's groups are kept intact until crossover point X, remaining tasks are added with π_2's precedences). Third, the resulting solution is repaired if it is invalid (6) with some probability $RepairRate$. The repair procedure tries to make an invalid solution into a valid one while retaining some of its characteristics, that is, the order prescribed by the solution. For the 1-P problem, iterating over tasks in order, any task that is involved in a conflicting precedence constraint is pushed back until it can be scheduled. For the JS-P problem, the repair procedure works in the following way: groups at the same index on every machine are merged into one big "mega-group" of tasks, then, for each group, for each occurrence of a job's task in the group, the job's next task is inserted into the corresponding machine current group. After a "mega-group" is fully scheduled, groups on all machines are closed, before scheduling the next mega-group. Because the tasks of a job are inserted in increasing order, this repair method can never lead to a fully infeasible solution. Fourth, the solution is mutated (9) with some probability $MutateRate$, performing a random move from the neighborhoods described in 5.2.1. Fifth it is educated (12) with some probability $EducateRate$, and added to the population. The education procedure repeatedly applies the first improving "Group insert" move until no more improving moves are found, or up to a maximum of 100 moves. Finally in the sixth step, if the size of the population is more than $PopulationLimit$, the worst half of the population is removed. Steps 2 to 6 are repeated until the time limit is reached or until the algorithm stagnates too much.

AG's meta-parameters were set empirically using grid testing. The implemented parameters are $(PopSize, PopLimit, MutateRate, EducationRate, RepairRate) = (10, 20, 0.5, 0.01, 0.5)$ for both problems.

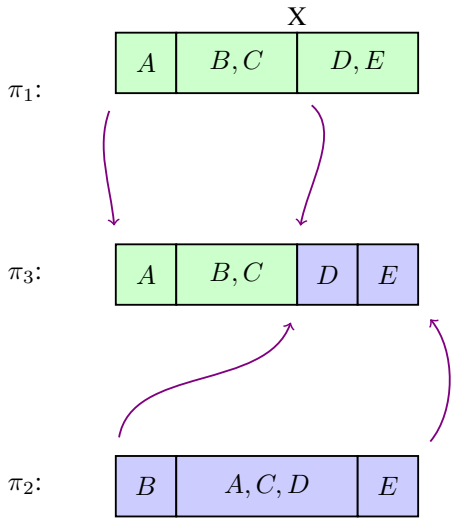


Fig. 8 The G-SEQ 1 point crossover

6 J-SEQ solution approaches

Literature review on the stochastic job-shop scheduling reveals that local search or metaheuristics approaches are mostly used (Ghasemi, Ashoori, & Heavey, 2021; Gu, Gu, & Gu, 2009; Hao, Lin, Gen, & Ohno, 2013; Horng & Lin, 2015). In (Akker, Blokland, & Hoogeveen, 2013), local search approaches that use simulation on a discrete scenario set are shown to outperform approaches that use surrogate measures by replacing the stochastic parameters either by some distribution percentile, or by some weighted expected value.

In line with these findings, we designed a genetic algorithm GA-J-SEQ and a tabu search method TAB-J-SEQ based on the one used for the G-SEQ scheme presented in sections 5.2.1 and 5.2.3, respectively, while restricting the search space to J-SEQ. The algorithm remains essentially the same, except for the absence of some neighborhoods (inserting a task in an other group, merging groups, splitting groups) .

We also use an exact approach in the form of a Constraint Programming model for the J-SEQ scheme: CP-J-SEQ. The model is directly taken from IBM CP Optimizer's *"stochastic jobshop"* example, which uses the **IloSameSequence()** global constraint to enforce sequence unity across all scenarios (see "CP-Optimizer 20.1.0 User Manual").

7 Experiments

7.1 Instances

The methods have been tested on a custom benchmark of instances generated using the following parameters:

- N The number of jobs
- M The number of machines (only for the jobshop problem).
- S The number of scenarios drawn for training
- Δ The variability of the release dates for 1-P and of the processing times for JS-P around a baseline value for each scenario .
- Π The density of the precedence graph (only for the single machine problem). This density is measured as the ratio of the number of precedence constraints over the maximum possible number of precedence constraints.

We first define a default parameter set ($N = 100$, $S = 25$, $\Delta = 0.3$, $\pi = 0.01$ for the 1-P and $N = 10$, $M = 10$, $S = 25$, $\Delta = 0.5$ for the JS-P). We chose these parameters such that they allow to observe the limits where our approach is useful. Then, in order to describe the relative variations in performance of the evaluated methods depending on the instance parameters, we generate 3 instance sets for the 1-P and 4 instances sets for the JS-P by keeping the default values for all parameters except for one that varies inside a given range.

Instance sets 1-P-N and JS-P-NM, contain instances of varying numbers of tasks (jobs/machine for the JS-P). Instance sets 1-P-S and JS-P-S gather instances of varying numbers of scenarios,. Instance sets 1-P-Δ and JS-P-Δ contain instances of varying scenario variability. Instance 1-P-Π contain instances of varying precedence constraints density. Note that inside each set 5 instances are randomly generated with the same set of parameters. In particular, in each set there are exactly 5 instances having the default parameter values. It follows that there are 20 instances having exactly the default parameter values for the 1-P and 15 instances having the default parameter values for the JS-P. These instances are gathered into sets 1-P-A and JS-P-A, respectively. Tables 1 and 2 give a summary of the instance set characteristics.

Inside each set, here is how the 5 instances are generated for each fixed set of parameters: 5 baseline scenarios are first generated in the following way: for each task, we chose a random base duration $p_i \in [50, 100]$, a random release date $r_i \in [0, \sum P/2]$, and a random due date $d_i \in [\sum P/2, \sum P]$. Precedence constraints are generated for each baseline scenario as follows: starting from an empty precedence set E, a random precedence constraint is added, using the lexicographical order to ensure feasibility, and the transitive closure of the matrix is performed. At each step, if the precedence density reaches or exceeds the desired density Π, the process stops.
Each baseline scenario is associated with a probability density: for each uncertain release date, a normal law $\mathcal{N}(r_i, r_i * \Delta)$ is used and the S scenarios from the training set as well as the 1000 scenarios of the test set are drawn from each baseline scenario. The

Set	N	Π	S	Δ	#inst.
1-P-N	[10, 20, 50, 100, 150, 200]	0.01	25	0.3	30
1-P-Π	100	[0, 0.001, 0.01, 0.05, 0.1, 0.2]	30	0.3	25
1-P-S	100	0.01	[2, 5, 10, 15, 20, 25, 35, 50, 100]	0.3	45
1-P-Δ	100	0.01	25	[0, 0.1, 0.3, 0.5, 0.7, 1]	30
1-P-A	100	0.01	25	0.3	20

Table 1: Instance sets for the single machine problem

Set	N	M	S	Δ	#inst.
JS-P-NM	[5,10,20]	[2,5,10]	25	0.5	45
JS-P-S	10	10	[2, 5, 10, 15, 20, 25, 35, 50, 100]	0.5	45
JS-P-Δ	10	10	25	[0, 0.1, 0.3, 0.5, 0.7, 1]	30
JS-P-A	10	10	25	0.5	15

Table 2: Instance sets for the jobshop problem

training set is used as the input scenario list for all methods while the solution obtained by each method is evaluated on the test set.

The instances are created for the jobshop problem in a similar fashion. For each fixed set of parameter in JS-P-NM, JS-P-S and JS-P-Δ, 5 baseline scenarios are generated while 15 baseline scenarios are generated for set JS-P-A. In each baseline scenario the duration of each task is generated randomly in $p_i^j \in [50, 100]$. Variations around the baseline duration are generated using an exponential law $p_i^j + \mathcal{E}(\Delta.p_i^j)$.

7.2 Evaluated methods

All instances are solved using the training scenario set by the different methods within a time limit set to 2 hours, for the 4 described objectives and with different parameter combinations (WC$\in\{$`false`,`true`$\}$, TPI$\in\{$`false`, `true`$\}$ and TINIT either equal to 0 when the method is not used in the warm start mode, or to a value in minutes for warm start. Table 3 gives a summary of the configurations that were tested on the described benchmark. C1 corresponds to the methods without warm-start. C2 evaluates the warm-start methods (indicated with a '*') for different starting times. C3 evaluates the same methods for starting time at 90 minutes, with or without parameters WC and

TPI (only in the case of JS-P).
Note that we also use the FIFO fully reactive heuristic as a comparison point, which can be assimilated to a single, fully permutable G-SEQ solution: the methods makes no anticipation and only react to the realized scenario by applying the FIFO priority rule. A total of 43 method variants are compared (37 for 1-P).

Conf.(#)	X-SEQ Method	Parameter values		
		WC	TPI (JS-P)	TINIT
C1 (7)	{CP-G,CP-J,AG-G,AG-J, TAB-G,TAB-J,FIFO-G}	`false`	`true`	0
C2 (24)	{AG-G*,EW-G*,TAB-G*}	`false`	`true`	{1,2,5,10,30 60, 90, 110}
C3 (12)	{AG-G*,EW-G*,TAB-G*}	{`true`,`false`}	{`true`,`false`}	90

Table 3: Evaluated Methods for both 1-P and JS-P

Note that we define a default method parameter configuration used for all G-SEQ methods unless stated otherwise to ¬WC and TPI, INIT at 90 minutes. The TPI parameter is set to `false` for 1-P.

7.3 Performance criteria

The methods are analysed and compared using the following metrics:

- **Score** is the average objective function value obtained by the method over the training set of instances at the maximum cutoff time of 2 hours.
- **Generalized score** is the average objective function value obtained by the solution obtained by the method during the training phase on the test set of instances. When $\sigma = max$, we use the 90% quantile instead of the actual maximum score to evaluate the solutions on the testing set.
- **Performance** (or **generalized performance**) is computed as the ratio of the best found (generalized) score among all methods to the (generalized) score of the evaluated method on an instance.
- **Generalization loss** is computed as the ratio of generalized score over the score.

Results are averaged across all 4 objectives. When investigating the impact of a parameter, methods are run on the set that make the considered parameter vary (e.g set 1-P-N when investigating the variation of N on the one machine problem). When investigating the impact of objectives, methods are run on the A sets.

8 Results

All methods were run on a single worker (Xeon E5-2695 v4 @ 2.10GHz processor). The CP models were solved by IBM CP Optimizer 20.1. For warm start heuristics

(indicated by "*"), the output of the J-SEQ CP model after TINIT seconds was used as starting point for the heuristic for the remaining time.
Note that the warm start heuristics were implemented using Python, while CP methods used the python API of CPO.
Tables 4, 5, 6, 7, 8, 9 and 10 display the generalized scores of a selection of methods with varying instance parameters. Tables 11,12,13,14,15,16, and 17 display the impact of the WC method parameter in function of instance parameters, and tables 18 and 19 displays the impact of TPI method parameter for different values of Δ (only for the job-shop problem). The following sections provide a detailed analysis on these results.

8.1 Comparing the J-SEQ methods

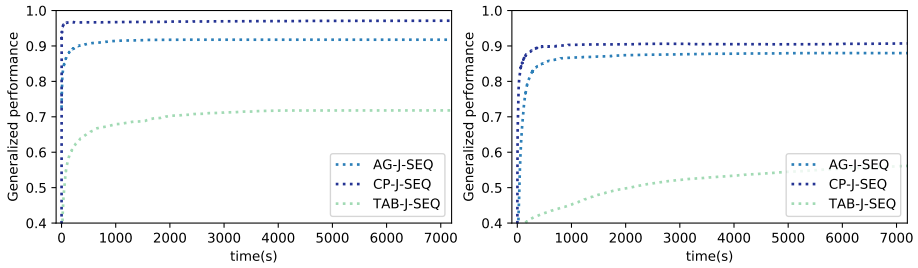
Tables 4, 5, 6, 7, 8, 9 and 10 allow us to compare the generalized performance of the J-SEQ CP solver with the GA and Tabu algorithms restricted to J-SEQ solutions (columns CP-J-SEQ, AG-J-SEQ and TAB-J-SEQ). Figure 9 summarizes the evolution over time of the generalized score of the three methods.

We can see from the Figure 9 and the different tables that the CP J-SEQ method performs overall better than the GA J-SEQ and TAB J-SEQ methods, both on the 1-P and JS-P problems. This performance gap likely does not come from the methods implementation, but rather to the lack of diversification of the methods. Indeed, figure 9 shows that they seem to on average reach some local optimum before the cutoff time.
The performance gap seems rather insensitive to the scenario variability (Tables 4 and 5) but is particularly noticed when the instance size increases (Tables 6 and 7), which emphasises CP optimizer's performances on these problems. We also notice that the GA-J-SEQ method performs much better than the TAB-J-SEQ method on average. Distinguishing the objectives reveal that for 1-P, The CP-J-SEQ method is really more effective for the L_{max} objective, while things are more mitigated for the $\sum C_i$ objective (Table 8). For the JS-P however, the CP-J-SEQ method is consistently superior for all objectives (Table 9). When precedence density increases, the gap between GA-J-SEQ and CP-J-SEQ gets narrow but still in favor of the latter for high densities (Table 10).

Therefore CP-J-SEQ is selected in the remaining as the reference two-phase method based on job sequences.

8.2 Comparing the CP-J-SEQ and CP-G-SEQ solvers

Tables 4, 5, 6, 7, 8, 9 and 10 also allows us to compare the performances of G-SEQ and J-SEQ CP methods relatively to each others. The data shows that overall the CP-J-SEQ method performs much better than it's G-SEQ counterpart. The exception being on the 1-P problem, on very small instances, where the G-SEQ solver is able to find better solutions (see columns CP-G-SEQ and CP-J-SEQ in table 6 for $N = 10$). This is consistent with the theory as the G-SEQ problem is solved to optimality, the G-SEQ solutions should always be at least as good as the best J-SEQ solutions. This is not reproduced for the JS-P as either the problem could not be solved to optimality (even for some small instances), or the instances are so small that there is little gain to be made (see columns CP-G-SEQ and CP-J-SEQ in Table 7). We also notice in Tables 4 and 5 that when there is more variability among scenarios (Δ gets large) there is a point at



(a) JS-P-A **(b)** 1P-A

Fig. 9 Average generalized performance over time of J-SEQ methods for all objectives on the A sets

which the CP-G-SEQ solver performs better than the J-SEQ solver ($\Delta = 0.5$ for the 1-P and $\Delta = 0.7$ for the JS-P). As shown by the good performance at the same points of the FIFO method, this it is due to the fact that when there is too much variability, the purely reactive method performs better. Indeed, the CP-G-SEQ method usually starts with a very flexible solution and rarely manages to improve it by reducing the number of groups. Hence, another interest of the group approach lies in its ability to detect an excessive variability in the training scenarios that makes anticipation useless.

Considering the different objectives (Tables 8 and 9), the CP-G-SEQ solver is outperformed by the CP-J-SEQ solver except for the $max \sum C_i$ objective where CP-G-SEQ is significantly better both for 1-P and JS-P. Also for this objective, the pure FIFO heuristic performs remarkably well. This underlines a particular difficulty in producing proactive schedules for the robust sum-of-completion-time problem.

For the 1-P, the density of the precedence constraints (Table 10) has a high impact on the relative performance of CP-G-SEQ and CP-J-SEQ: when the density is close to 0, CP-G-SEQ outperforms GP-J-SEQ but the performance of CP-G-SEQ monotonely decreases with the density while the reverse is true for CP-J-SEQ. Section 8.7 makes an attempt to explain this behavior.

As a synthesis of this comparison, despite the theoretical dominance of the G-SEQ model over the J-SEQ model, the CP-G-SEQ solvers manages to get better results than the CP-J-SEQ solver only in a few cases. More research on improving CP models for group sequences is needed.

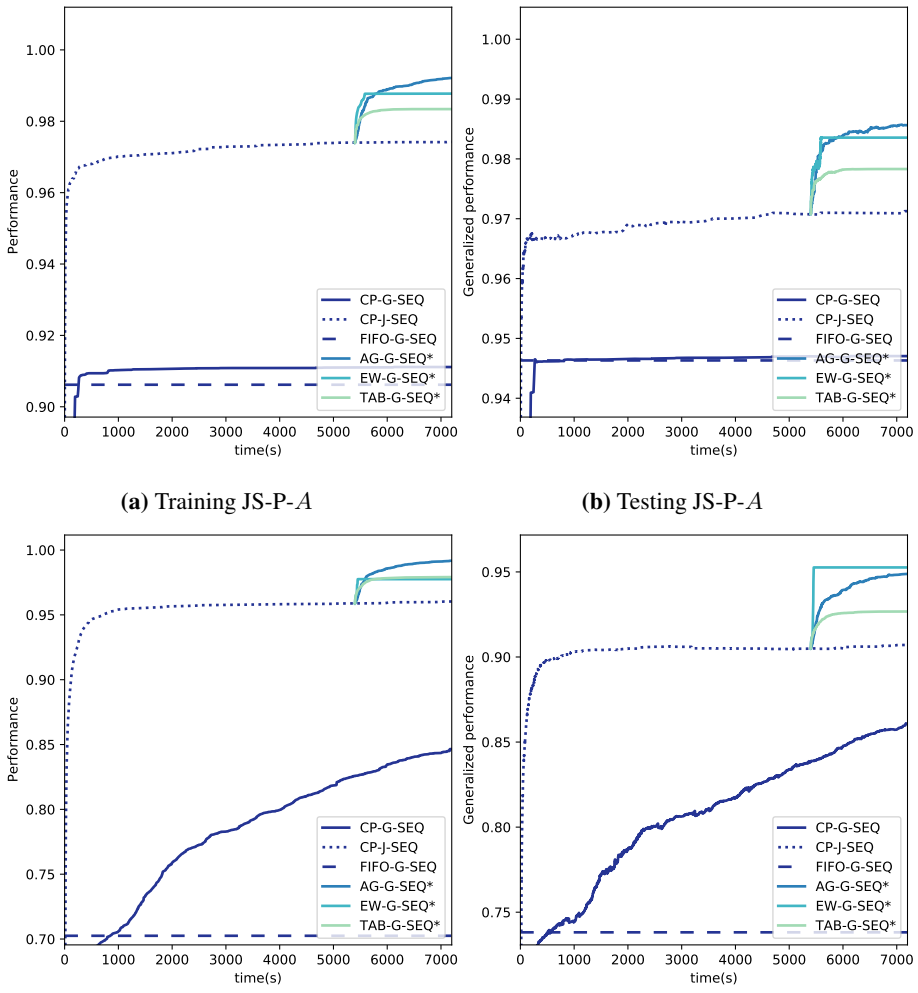
8.3 Comparing the CP-J-SEQ and the warm start G-SEQ methods with standard parameters

Figure 10 shows the average behavior of the different warm start G-SEQ methods with standard parameters and of the CP-J-SEQ method on the training and test A instances. The results on the test A set are also detailed in Tables 4, 5, 6, 7, 8, 9 and 10.

We can observe in Figure 10 that starting from a good J-SEQ solution computed by the J-SEQ CP solver at TINIT=90, the warm start G-SEQ heuristics (AG-G-SEQ*, EW-G-SEQ* and TAB-G-SEQ*) rapidly boost the solutions' performances by adding

flexibility and significantly improving the score, which is also observed in the testing phase. Tables 4 and 5 show that using the warm start G-SEQ methods, we are able to improve marginally the solution quality of the J-SEQ methods even when Δ is low, but much more noticeably when the variability increases. However, for the 1-P, when variability is too high, the warm start methods are not able to provide a better solution than the purely reactive approach. The impact of variability reveals that TAB-G-SEQ* is almost always dominated by AG-G-SEQ* and EW-G-SEQ* while EW-G-SEQ* is competitive with AG-G-SEQ*except for large variability where its performance collapses. The problem size (Tables 6 and 7) does not seem to influence significantly the relative performance of the warm start methods. On the contrary, Tables 8 show that the objective function has an impact on the warm start method performance on the 1-P: the GA-G-SEQ is the best over CP-J-SEQ and EW-G-SEQ for all objectives except $\max L_{\max}$ where EW-G-SEQ outperforms GA-G-SEQ and CP-J-SEQ. No such behavior is observed for the JS-P (Table 9). Finally Table 10 shows that the warm start solver performance is less deteriorated by the precedence constraint density increase than that of the CP-G-SEQ solver and they remain consistently better than CP-J-SEQ. This is to be expected as their solutions are built upon the sequence solutions provided by CP-J-SEQ.

It follows from the analysis of these results that the G-SEQ warm start methods offers a superior alternative to purely sequencing methods in the majority of cases.



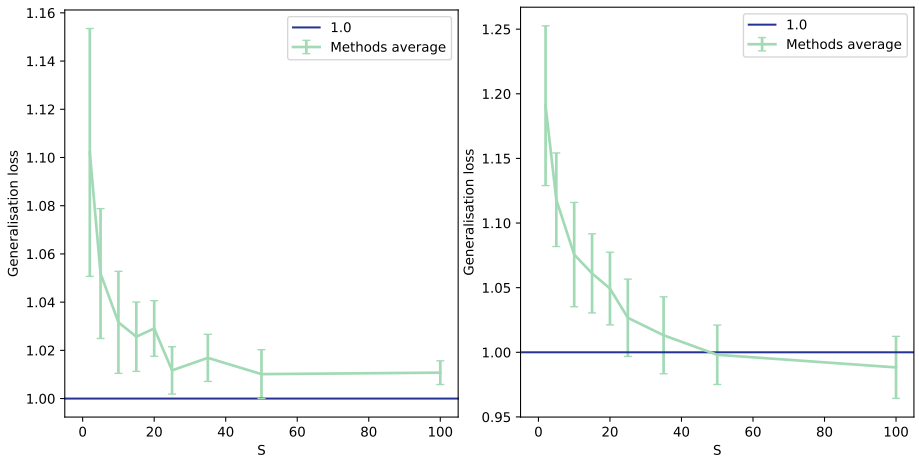
(a) Training JS-P-A **(b)** Testing JS-P-A

(c) Training 1-P-A **(d)** Testing 1-P-A

Fig. 10 Average performance and generalized performance over time for all objectives of the G-SEQ methods with default parameters on the A sets

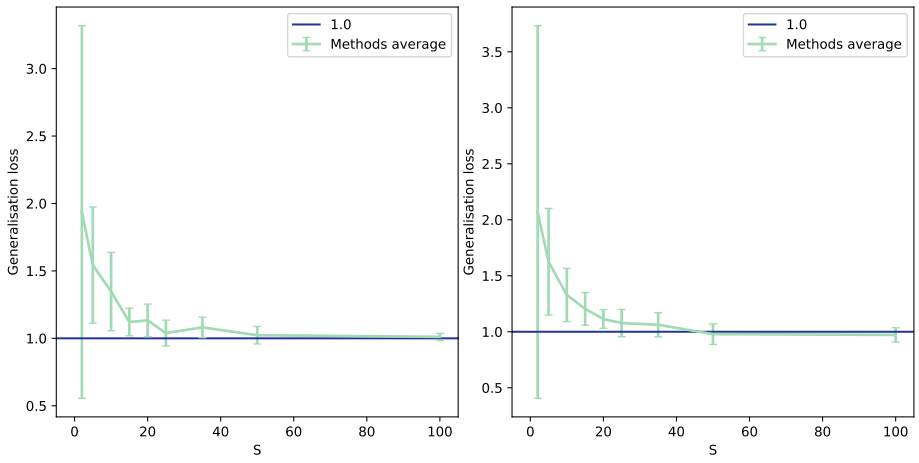
8.4 Generalisation score

Because solutions are computed using a training set of scenarios, one might worry that the computed solutions do not generalize well to the actual distribution of the data. Figure 11 shows that the average generalization loss reduces very quickly with the number of training scenarios for all methods, for both the 1-P and the JS-P.
Note that for the max objective, the generalisation loss seems to converge to a value smaller than 1. That is because even though ”max” scores are computed over all scenarios during training, when evaluating we use the 90th percentile.



(a) JS-P-S ($\sigma = avg$)

(b) JS-P-S ($\sigma = max$)



(c) 1P-S ($\sigma = avg$)

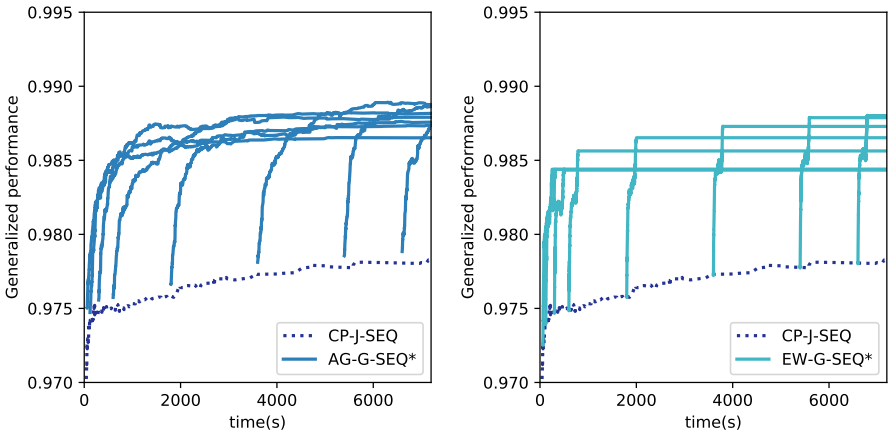
(d) 1P-S ($\sigma = max$)

Fig. 11 Impact of the number of training scenarios on the average generalisation loss of all methods with default parameters on the S sets for different objective aggregators

8.5 Impact of TINIT

Figure 12 compares generalized performance over time of the AG-G-SEQ and EW-G-SEQ methods starting from a CP-J-SEQ solution. The analysis shows that the EW method benefits greatly from starting after the J-SEQ solver reaches a plateau, while the AG method is able to diversify its solutions more effectively and is less sensitive to the quality of the given initial solution. For any TINIT value, the improvement brought over the J-SEQ solution is always significant. However, because it takes more time to reach a local optimum, launching it with too little remaining time leads to a smaller gain. When

dealing with limited time, even if there is no time for J-SEQ to reach a plateau, it might be beneficial to divide appropriately the available time and launch a warm start method.



(a) JS-P-A AG-G-SEQ*

(b) JS-P-A EW-G-SEQ*

(c) 1-P-A AG-G-SEQ*

(d) 1-P-A EW-G-SEQ*

Fig. 12 Impact of TINIT values for warm start methods with default parameters on the average generalized performance over time on the A sets

8.6 Impact of the ”worst-case” parameter

We next study the impact of the ”worst-case” (WC) parameter. That is, we try to assert whether it is more beneficial during the training phase to evaluate candidate G-SEQ solutions using the FIFO heuristic on training scenarios, or to evaluate them using the worst-case (WC) bound algorithm described in section 1. Table 11 and 12 describes the average generalized performances for different variability Δ. Results show that

the more variable the training scenarios are, the more preferable it is to use the FIFO heuristic during the optimization process. Table 13 shows that for the 1-P, less relative gain is made using the FIFO heuristic for larger instances. However, the instance size does not have so much impact on the JS-P. Table 17 also shows that more precedence constraints reduces the gain made by using the FIFO heuristic, due to more sequential solutions reducing the gap between both evaluation score. Interestingly, Table 15 shows that for $maxL_{max}$ objective on the 1-P, it is beneficial to use worst-case optimization.

8.7 Impact of tolerating partially invalid solutions.

We showed earlier that when using our heuristic to schedule tasks from a G-SEQ solution, a valid sequence would be generated as long as the G-SEQ is partially valid. However, tolerating partially valid solutions might hinder the search by modifying the search space.
Results on the jobshop problem (see Table 18) show that for most warm start heuristics it is better on average to tolerate partially valid solutions when Δ is high, and slightly worse when it is small. And Table 19 shows that the $\sum C_i$ objective benefits from TPI but C_{max} does not.

So, globally, tolerating partially invalid solutions for the JS-P appears beneficial when the variability increases. This could explain the already mentioned collapse of the CP-G-SEQ method when the density of precedence constraint increases for the 1-P (Table 10). Indeed as the TPI feature was not implemented for the one-machine problem, imposing that all job sequences represented by the G-SEQ are valid is maybe too restricting when there are many precedence constraints. This is why the flexibility brought by the pure FIFO heuristic pays more in this case.

9 Concluding remarks

In this work, new methods for computing G-SEQ solutions were introduced, namely an exact methods using constraint programming, and meta-heuristics (Genetic algorithm and tabu method) with mitigated results. Extensive experiments were run to characterize when and how computing G-SEQ could improve on a J-SEQ solution in limited time.
Results show that the G-SEQ approach is most effective when variability is moderate, while J-SEQ methods outperform them when variability is very low and the purely reactive FIFO method outperforms the other methods when scenarios become very unpredictable. Among the warm start heuristics, the simpler greedy heuristic (EW method) is most of the time very effective.
Because the greedy heuristic can only merge groups together, the explored space is very restricted, and the fact that this method is one of, if not the best method shows how difficult it is to search the G-SEQ solution space (at least using the proposed neighborhoods). However, even with current search methods, results show that some improvement over regular J-SEQ solutions can be made in almost all cases provided the available computation time is adequately divided between the J-SEQ solver, and the G-SEQ warm start heuristics.
We identify several areas for further research to improve the presented approach and

expand its scope. The first point of improvement lies in the quality of the G-SEQ solutions computed. Exact CP approaches could benefit from the creation of dedicated constraint propagation algorithm or branching heuristics to accelerate the search. Other heuristic approaches can also be investigated, such as large neighborhood searche using constraint programming for example. The theoretical results that were established in this paper could help in exploring the search space. The second point of improvement lies in the capabilities of the presented framework. Indeed, an ideal two stage approach would map for each scenario its optimal corresponding J-SEQ. Our approach is limited both by the representative capabilities of G-SEQs (they cannot represent arbitrary sets of sequences), and by the matching capabilities of the FIFO heuristics. As such, future work should investigate the usage of other flexible structures as first-stage decision frameworks (e.g. partial orders, AND/OR-trees, decision diagrams) and of other second-stage decision policies (possibly using more information than FIFO). Finally, the method could be extended to more general scheduling problems such as the RCPSP.

References

Akker, M. v. d., Blokland, K. v., & Hoogeveen, H. (2013). Finding robust solutions for the stochastic job shop scheduling problem by including simulation in local search. In *International symposium on experimental algorithms* (pp. 402–413). doi: https://doi.org/10.1007/978-3-642-38527-8_35

Artigues, C., Billaut, J.-C., & Esswein, C. (2005, September). Maximization of solution flexibility for robust shop scheduling. *European Journal of Operational Research*, *165*(2), 314–328. doi: https://doi.org/10.1016/j.ejor.2004.04.004

Artigues, C., Jean-Charles, B., Cheref, A., Mebarki, N., & Yahouni, Z. (2016). Robust Machine Scheduling Based on Group of Permutable Jobs. In D. M, Z. C, & G. E (Eds.), *Robustness Analysis in Decision Aiding* (Vol. 241, pp. 191–220). Springer. doi: https://doi.org/10.1007/978-3-319-33121-8_9

Aytug, H., Lawley, M. A., McKay, K., Mohan, S., & Uzsoy, R. (2005). Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, *161*(1), 86–110. doi: https://doi.org/10.1016/j.ejor.2003.08.027

Bachtler, O., Krumke, S. O., & Le, H. M. (2020). Robust single machine makespan scheduling with release date uncertainty. *Operations Research Letters*, *48*(6), 816–819.

Balas, E. (1969). Machine sequencing via disjunctive graphs: an implicit enumeration algorithm. *Operations research*, *17*(6), 941–957. doi: https://doi.org/10.1287/opre.17.6.941

Billaut, J.-C., & Roubellat, F. (1996). A new method for workshop real time scheduling. *International Journal of Production Research*, *34*(6), 1555–1579.

Cardin, O., Mebarki, N., & Pinot, G. (2013, November). A study of the robustness of the group scheduling method using an emulation of a complex FMS. *International Journal of Production Economics*, *146*(1), 199–207. doi: https://doi.org/10.1016/j.ijpe.2013.06.023

Cheref, A., Artigues, C., & Billaut, J.-C. (2016a). A new robust approach for a production scheduling and delivery routing problem. *IFAC-PapersOnLine*, *49*(12), 886–891. doi: https://doi.org/10.1016/j.ifacol.2016.07.887

Cheref, A., Artigues, C., & Billaut, J.-C. (2016b). *Online recoverable robustness based on groups of permutable jobs for integrated production scheduling and delivery routing.* (Research report LAAS-CNRS https://hal.archives-ouvertes.fr/hal-01351496)

Daniels, R. L., & Kouvelis, P. (1995). Robust scheduling to hedge against processing time uncertainty in single-stage production. *Management science*, *41*(2), 363–376. doi: https://doi.org/10.1287/mnsc.41.2.363

Davari, M., & Demeulemeester, E. (2019, April). The proactive and reactive resource-constrained project scheduling problem. *Journal of Scheduling*, *22*. doi: https://doi.org/10.1007/s10951-017-0553-x

Erschler, J., & Roubellat, F. (1989). An Approach to Solve Workshop Real Time Scheduling Problems. In S. Y. Nof & C. L. Moodie (Eds.), *Advanced Information Technologies for Industrial Material Flow Systems* (pp. 651–679). Berlin, Heidelberg: Springer. doi: https://doi.org/10.1007/978-3-642-74575-1_27

Esswein, C. (2003). *Un apport de flexibilité séquentielle pour l'ordonnancement robuste* (These de doctorat, Tours). Retrieved from http://www.theses.fr/2003TOUR4022

Ghasemi, A., Ashoori, A., & Heavey, C. (2021). Evolutionary learning based simulation optimization for stochastic job shop scheduling problems. *Applied Soft Computing*, *106*, 107309. doi: https://doi.org/10.1016/j.asoc.2021.107309

Goldratt, E. (1997). *Critical chain*. North River Press. doi: https://doi.org/doi.org/10.4324/9781351218986

Gu, J., Gu, X., & Gu, M. (2009). A novel parallel quantum genetic algorithm for stochastic job shop scheduling. *Journal of Mathematical Analysis and Applications*, *355*(1), 63–81. doi: https://doi.org/10.1016/j.jmaa.2008.12.065

Hao, X., Lin, L., Gen, M., & Ohno, K. (2013). Effective estimation of distribution algorithm for stochastic job shop scheduling problem. *Procedia Computer Science*, *20*, 102–107. doi: https://doi.org/10.1016/j.procs.2013.09.246

Herroelen, W., & Leus, R. (2005, September). Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, *165*(2), 289–306. doi: https://doi.org/10.1016/j.ejor.2004.04.002

Horng, S.-C., & Lin, S.-S. (2015). Integrating ant colony system and ordinal optimization for solving stochastic job shop scheduling problem. In *2015 6th international conference on intelligent systems, modelling and simulation* (pp. 70–75). doi: https://doi.org/10.1109/isms.2015.9

Kouvelis, P., & Yu, G. (1997). *Robust Discrete Optimization and Its Applications* (Vol. 14). Boston, MA: Springer US. doi: https://doi.org/10.1007/978-1-4757-2620-6

Li, Z., & Ierapetritou, M. (2008). Process scheduling under uncertainty: Review and

challenges. *Computers & Chemical Engineering*, *32*(4-5), 715–727.

Möhring, R. H., Radermacher, F. J., & Weiss, G. (1984). Stochastic scheduling problems i—general strategies. *Zeitschrift für Operations Research*, *28*(7), 193–260. doi: https://doi.org/10.1007/bf01919323

Pinedo, M., & Schrage, L. (1982). Stochastic shop scheduling: A survey. In *Deterministic and stochastic scheduling* (pp. 181–196). Springer. doi: https://doi.org/10.1007/978-94-009-7801-0_9

Pinedo, M. L. (2012). *Scheduling* (Vol. 29). Springer.

Pinot, G., Cardin, O., & Mebarki, N. (2007, October). A study on the group sequencing method in regards with transportation in an industrial FMS. In *2007 IEEE International Conference on Systems, Man and Cybernetics* (pp. 151–156). (ISSN: 1062-922X) doi: https://doi.org/10.1109/ICSMC.2007.4414227

Roy, B., & Sussmann, B. (1964). Les problemes d'ordonnancement avec contraintes disjonctives. *Note ds*, *9*.

Vepsalainen, A. P., & Morton, T. E. (1987). Priority rules for job shops with weighted tardiness costs. *Management science*, *33*(8), 1035–1047. doi: https://doi.org/10.1287/mnsc.33.8.1035

Wang, B., Wang, X., Lan, F., & Pan, Q. (2018). A hybrid local-search algorithm for robust job-shop scheduling under scenarios. *Applied Soft Computing*, *62*, 259–271. doi: https://doi.org/10.1016/j.asoc.2017.10.020

Wang, B., Wang, X., & Xie, H. (2019). Bad-scenario-set robust scheduling for a job shop to hedge against processing time uncertainty. *International Journal of Production Research*, *57*(10), 3168–3185. doi: https://doi.org/10.1080/00207543.2018.1555650

Wu, C. W., Brown, K. N., & Beck, J. C. (2005). Scheduling with uncertain release dates. *AICS'05*, 397.

Wu, S. D., Byeon, E.-S., & Storer, R. H. (1999). A graph-theoretic decomposition of the job shop scheduling problem to achieve scheduling robustness. *Operations research*, *47*(1), 113–124. doi: https://doi.org/10.1287/opre.47.1.113

A Appendix: results of all methods

Δ	AG-G-SEQ	AG-J-SEQ	CP-G-SEQ	CP-J-SEQ	FIFO-G-SEQ	TAB-G-SEQ	TAB-J-SEQ
0.0000	0.9828	0.9931	0.8406	0.9997	0.4613	0.3950	0.4242
0.1000	0.9467	0.8868	0.7496	0.9124	0.5013	0.4340	0.4537
0.3000	0.9152	0.8796	0.8600	0.9164	0.7300	0.5631	0.5533
0.5000	0.8461	0.7880	0.8133	0.8024	0.9239	0.5876	0.5807
0.7000	0.8104	0.7397	0.7801	0.7527	0.9852	0.5833	0.5780
1.0000	0.7284	0.6721	0.7500	0.6919	1.0000	0.5921	0.5784

Δ	AG-G-SEQ*	EW-G-SEQ*	TAB-G-SEQ*
0.0000	0.9998	0.9998	0.9997
0.1000	0.9424	0.9836	0.9298
0.3000	0.9524	0.9487	0.9319
0.5000	0.8345	0.8349	0.8207
0.7000	0.7874	0.7762	0.7738
1.0000	0.7244	0.7123	0.7110

Table 4: Impact of scenario variability on the average generalized performance of the methods with default parameters on the 1-P-Δ set

Δ	AG-G-SEQ	AG-J-SEQ	CP-G-SEQ	CP-J-SEQ	FIFO-G-SEQ	TAB-G-SEQ	TAB-J-SEQ
0.0000	0.9594	0.8984	0.9579	1.0000	0.8689	0.9396	0.5415
0.1000	0.9672	0.9055	0.8757	0.9974	0.8697	0.9389	0.6536
0.3000	0.9738	0.9189	0.9036	0.9841	0.9139	0.9504	0.6910
0.5000	0.9734	0.9120	0.9460	0.9687	0.9446	0.9631	0.7171
0.7000	0.9828	0.9083	0.9607	0.9605	0.9626	0.9738	0.7049
1.0000	0.9872	0.8956	0.9721	0.9422	0.9721	0.9789	0.7135

Δ	AG-G-SEQ*	EW-G-SEQ*	TAB-G-SEQ*
0.0000	1.0000	1.0000	1.0000
0.1000	0.9975	0.9921	0.9975
0.3000	0.9882	0.9841	0.9863
0.5000	0.9828	0.9823	0.9756
0.7000	0.9802	0.9755	0.9685
1.0000	0.9803	0.9689	0.9529

Table 5: Impact of scenario variability on the average generalized performance of the methods with default parameters on the JS-P-Δ set

N	AG-G-SEQ	AG-J-SEQ	CP-G-SEQ	CP-J-SEQ	FIFO-G-SEQ	TAB-G-SEQ	TAB-J-SEQ
10	0.9812	0.9075	0.9802	0.9142	0.6853	0.9665	0.9086
20	0.9624	0.8351	0.9497	0.8380	0.6500	0.9138	0.8031
50	0.9230	0.8876	0.8644	0.9229	0.5786	0.8522	0.8059
100	0.9366	0.8883	0.8714	0.9151	0.7297	0.5991	0.5764
150	0.8455	0.8829	0.7308	0.9305	0.7714	0.4419	0.4585
200	0.6829	0.8672	0.6754	0.9128	0.7980	0.4123	0.4234

N	AG-G-SEQ*	EW-G-SEQ*	TAB-G-SEQ*
10	0.9857	0.9919	0.9836
20	0.9579	0.9549	0.9061
50	0.9662	0.9635	0.9369
100	0.9525	0.9654	0.9319
150	0.9380	0.9247	0.9137
200	0.9226	0.9368	0.9180

Table 6: Impact of instance size on the average generalized performance of the methods with default parameters on the 1-P-N set

N	M	AG-G-SEQ	AG-J-SEQ	CP-G-SEQ	CP-J-SEQ	FIFO-G-SEQ	TAB-G-SEQ	TAB-J-SEQ
5	2	0.9952	0.9941	0.9917	0.9997	0.9463	0.9909	0.9543
	5	0.9844	0.9553	0.9856	0.9865	0.9520	0.9789	0.8918
	10	0.9916	0.9579	0.9851	0.9700	0.9785	0.9873	0.8006
10	2	0.9937	0.9767	0.9842	0.9928	0.9113	0.9808	0.9637
	5	0.9817	0.9549	0.9305	0.9848	0.9462	0.9742	0.8791
	10	0.9819	0.9198	0.9543	0.9745	0.9528	0.9742	0.7095
20	2	0.9898	0.9709	0.8193	0.9966	0.8711	0.9771	0.9590
	5	0.9835	0.9443	0.8598	0.9880	0.9009	0.9595	0.8705
	10	0.9714	0.9072	0.9399	0.9825	0.9525	0.9753	0.5795

N	M	AG-G-SEQ*	EW-G-SEQ*	TAB-G-SEQ*
5	2	0.9997	0.9998	0.9997
	5	0.9917	0.9908	0.9873
	10	0.9906	0.9857	0.9806
10	2	0.9965	0.9970	0.9928
	5	0.9897	0.9917	0.9852
	10	0.9878	0.9841	0.9809
20	2	0.9983	0.9983	0.9964
	5	0.9911	0.9915	0.9898
	10	0.9863	0.9888	0.9857

Table 7: Impact of instance size on the average generalized performance of the methods with default parameters on the JS-P-NM set

σ	γ	AG-G-SEQ	AG-J-SEQ	CP-G-SEQ	CP-J-SEQ	FIFO-G-SEQ	TAB-G-SEQ	TAB-J-SEQ
avg	L_{max}	0.9043	0.8351	0.6450	0.8836	0.3317	0.3743	0.3417
	$\sum C_i$	0.9891	0.9289	0.9662	0.9265	0.9736	0.8203	0.8390
max	L_{max}	0.8854	0.8626	0.8630	0.9229	0.6477	0.3819	0.3957
	$\sum C_i$	0.9584	0.8926	0.9694	0.8953	0.9997	0.7286	0.6688

σ	γ	AG-G-SEQ*	EW-G-SEQ*	TAB-G-SEQ*
avg	L_{max}	0.9593	0.9567	0.9166
	$\sum C_i$	0.9813	0.9566	0.9618
max	L_{max}	0.9214	0.9818	0.9229
	$\sum C_i$	0.9332	0.9154	0.9056

Table 8: Impact of the objective on the average generalized performance of the methods with default parameters on the 1-P-A set

σ	γ	AG-G-SEQ	AG-J-SEQ	CP-G-SEQ	CP-J-SEQ	FIFO-G-SEQ	TAB-G-SEQ	TAB-J-SEQ
avg	C_{max}	0.9851	0.9432	0.9291	0.9834	0.9251	0.9736	0.8594
	$\sum C_i$	0.9925	0.9198	0.9473	0.9664	0.9475	0.9788	0.7984
max	C_{max}	0.9555	0.9214	0.9406	0.9772	0.9413	0.9384	0.5632
	$\sum C_i$	0.9817	0.8862	0.9712	0.9581	0.9714	0.9736	0.6511

σ	γ	AG-G-SEQ*	EW-G-SEQ*	TAB-G-SEQ*
avg	C_{max}	0.9932	0.9902	0.9916
	$\sum C_i$	0.9887	0.9824	0.9799
max	C_{max}	0.9789	0.9821	0.9782
	$\sum C_i$	0.9818	0.9795	0.9635

Table 9: Impact of the objective on the average generalized performance of the methods with default parameters on the JS-P-A set

Π	AG-G-SEQ	AG-J-SEQ	CP-G-SEQ	CP-J-SEQ	FIFO-G-SEQ	TAB-G-SEQ	TAB-J-SEQ
0.0000	0.9201	0.8272	0.8590	0.8369	0.6246	0.6385	0.5033
0.0010	0.9183	0.8209	0.8567	0.8586	0.6559	0.6767	0.5507
0.0100	0.9522	0.8726	0.8496	0.8953	0.7545	0.5796	0.5539
0.0500	0.9079	0.8933	0.7137	0.8969	0.8396	0.6083	0.6196
0.1000	0.8907	0.8886	0.7120	0.8975	0.8629	0.6543	0.6660
0.2000	0.9196	0.9178	0.7302	0.9189	0.8909	0.7103	0.7409

Π	AG-G-SEQ*	EW-G-SEQ*	TAB-G-SEQ*
0.0000	0.8866	0.9467	0.8497
0.0010	0.9070	0.9476	0.8765
0.0100	0.9515	0.9510	0.9256
0.0500	0.9222	0.9148	0.9154
0.1000	0.9123	0.9137	0.9095
0.2000	0.9245	0.9294	0.9245

Table 10: Impact of precedence density on the average generalized performance of the methods with default parameters on the 1-P-Π set

B Appendix: WC results

solver	AG-G-SEQ*		EW-G-SEQ*		TAB-G-SEQ*	
WC	False	True	False	True	False	True
Δ						
0.0000	0.9998	0.9997	0.9998	0.9997	0.9997	0.9997
0.1000	0.9424	0.9126	0.9836	0.9380	0.9298	0.9126
0.3000	0.9524	0.9150	0.9487	0.9295	0.9319	0.9150
0.5000	0.8345	0.8032	0.8349	0.8139	0.8207	0.8032
0.7000	0.7874	0.7531	0.7762	0.7588	0.7738	0.7531
1.0000	0.7244	0.6913	0.7123	0.6980	0.7110	0.6915

Table 11: Impact of the WC parameter on the average generalized performance of warm-start methods for different scenario variabilities on the 1-P-Δ set

solver	AG-G-SEQ*		EW-G-SEQ*		TAB-G-SEQ*	
WC	False	True	False	True	False	True
Δ						
0.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
0.1000	0.9975	0.9974	0.9921	0.9979	0.9975	0.9974
0.3000	0.9882	0.9833	0.9841	0.9840	0.9863	0.9833
0.5000	0.9828	0.9678	0.9823	0.9691	0.9756	0.9678
0.7000	0.9802	0.9596	0.9755	0.9611	0.9685	0.9596
1.0000	0.9803	0.9422	0.9689	0.9438	0.9529	0.9422

Table 12: Impact of the WC parameter on the average generalized performance of warm-start methods for different scenario variabilities on the JS-P-Δ set

solver	AG-G-SEQ*		EW-G-SEQ*		TAB-G-SEQ*	
WC	False	True	False	True	False	True
N						
10	0.9857	0.9142	0.9919	0.9158	0.9836	0.9142
20	0.9579	0.8380	0.9549	0.8464	0.9061	0.8380
50	0.9662	0.9230	0.9635	0.9471	0.9369	0.9229
100	0.9525	0.9140	0.9654	0.9325	0.9319	0.9141
150	0.9380	0.9022	0.9247	0.9030	0.9137	0.9024
200	0.9226	0.9109	0.9368	0.9109	0.9180	0.9112

Table 13: Impact of the WC parameter on the average generalized performance of warm-start methods for different instance size on the 1-P-N set

	solver	AG-G-SEQ*		EW-G-SEQ*		TAB-G-SEQ*	
	WC	False	True	False	True	False	True
N	M						
5	2	0.9997	0.9997	0.9998	0.9998	0.9997	0.9997
	5	0.9917	0.9851	0.9908	0.9862	0.9873	0.9851
	10	0.9906	0.9700	0.9857	0.9713	0.9806	0.9700
10	2	0.9965	0.9928	0.9970	0.9972	0.9928	0.9928
	5	0.9897	0.9838	0.9917	0.9861	0.9852	0.9838
	10	0.9878	0.9746	0.9841	0.9761	0.9809	0.9746
20	2	0.9983	0.9964	0.9983	0.9982	0.9964	0.9964
	5	0.9911	0.9883	0.9915	0.9889	0.9898	0.9883
	10	0.9863	0.9819	0.9888	0.9829	0.9857	0.9818

Table 14: Impact of the WC parameter on the average generalized performance of warm-start methods for different instance size on the JS-P-NM set

	solver	AG-G-SEQ*		EW-G-SEQ*		TAB-G-SEQ*	
	WC	False	True	False	True	False	True
σ	γ						
avg	$L_{\max}$	0.9593	0.8757	0.9567	0.8868	0.9166	0.8757
	$\sum C_i$	0.9813	0.9262	0.9566	0.9262	0.9618	0.9262
max	$L_{\max}$	0.9214	0.9229	0.9818	0.9751	0.9229	0.9229
	$\sum C_i$	0.9332	0.8954	0.9154	0.8952	0.9056	0.8953

Table 15: Impact of the WC parameter on the average generalized performance of warm-start methods for different objectives on the 1-P-A set

	solver	AG-G-SEQ*		EW-G-SEQ*		TAB-G-SEQ*	
	WC	False	True	False	True	False	True
σ	γ						
avg	$C_{\max}$	0.9932	0.9823	0.9902	0.9824	0.9916	0.9823
	$\sum C_i$	0.9887	0.9664	0.9824	0.9664	0.9799	0.9664
max	$C_{\max}$	0.9789	0.9765	0.9821	0.9818	0.9782	0.9765
	$\sum C_i$	0.9818	0.9581	0.9795	0.9581	0.9635	0.9581

Table 16: Impact of the WC parameter on the average generalized performance of warm-start methods for different objectives on the JS-P-A set

solver	AG-G-SEQ*		EW-G-SEQ*		TAB-G-SEQ*	
WC	False	True	False	True	False	True
Π						
0.0000	0.8866	0.8391	0.9467	0.8654	0.8497	0.8389
0.0010	0.9070	0.8563	0.9476	0.8817	0.8765	0.8564
0.0100	0.9515	0.8953	0.9510	0.9070	0.9256	0.8953
0.0500	0.9222	0.8974	0.9148	0.9042	0.9154	0.8974
0.1000	0.9123	0.8974	0.9137	0.9018	0.9095	0.8976
0.2000	0.9245	0.9187	0.9294	0.9218	0.9245	0.9188

Table 17: Impact of the WC parameter on the average generalized performance of warm-start methods for different precedence densities on the 1-P-Π set

C Appendix: TPI results

solver	AG-G-SEQ*		EW-G-SEQ*		TAB-G-SEQ*	
TPI	False	True	False	True	False	True
Δ						
0.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
0.1000	0.9977	0.9975	0.9936	0.9921	0.9975	0.9975
0.3000	0.9893	0.9882	0.9850	0.9841	0.9863	0.9863
0.5000	0.9807	0.9828	0.9811	0.9823	0.9753	0.9756
0.7000	0.9743	0.9802	0.9760	0.9755	0.9683	0.9685
1.0000	0.9582	0.9803	0.9613	0.9689	0.9532	0.9529

Table 18: Impact of the TPI parameter on the average generalized performance of warm-start methods for different scenario variabilities on the JS-P-Δ set

	solver	AG-G-SEQ*		EW-G-SEQ*		TAB-G-SEQ*	
	TPI	False	True	False	True	False	True
σ	γ						
avg	$C_{\max}$	0.9940	0.9932	0.9915	0.9902	0.9916	0.9916
	$\sum C_i$	0.9812	0.9887	0.9797	0.9824	0.9798	0.9799
max	$C_{\max}$	0.9810	0.9789	0.9865	0.9821	0.9782	0.9782
	$\sum C_i$	0.9687	0.9818	0.9699	0.9795	0.9635	0.9635

Table 19: Impact of the TPI parameter on the average generalized performance of warm-start methods for different objectives on the JS-P-A set