An Efficient Constraint Programming Approach to Preemptive Job Shop Scheduling

Carla Juvin, Emmanuel Hebrard, Laurent Houssin, Pierre Lopez

An Efficient Constraint Programming Approach to Preemptive Job Shop Scheduling

Carla Juvin
LAAS-CNRS, Université de Toulouse, France

Emmanuel Hebrard
LAAS-CNRS, Université de Toulouse, France

Laurent Houssin
ISAE-SUPAERO, Université de Toulouse, France

Pierre Lopez
LAAS-CNRS, Université de Toulouse, France

Abstract

Constraint Programming has been widely, and very successfully, applied to scheduling problems. However, the focus has been on uninterruptible tasks, and preemptive scheduling problems are typically harder for existing constraint solvers. Indeed, one usually needs to represent all potential task interruptions thus introducing many variables and symmetrical or dominated choices.

In this paper, building on mostly known results, we observe that a large class of preemptive disjunctive scheduling problems do not require an explicit model of task interruptions. We then introduce a new constraint programming approach for this class of problems that significantly outperforms state-of-the-art dedicated approaches in our experimental results.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming; Computing methodologies → Planning and scheduling

Keywords and phrases Constraint Programming, Scheduling, Preemptive Resources

Digital Object Identifier 10.4230/LIPIcs.CP.2023.19

Supplementary Material *Software (Source Code)*: `https://github.com/ehebrard/Mistral-2.0.git`, archived at `swh:1:dir:90fe19df244134e5dfddd79360cd4b554fecabfb`

Funding *Emmanuel Hebrard*: ANR Project AD-Lib (ANR-22-CE23-0014).

Acknowledgements We would like to thank Claude-Guy Quimper for the advice and discussions while writing this paper.

1 Introduction

Many applications involve scheduling a set of tasks subject to resource constraints. Constraint programming (CP) techniques lead to significant advances in this domain and, conversely, some of the early work on the propagation of global constraints originated from scheduling applications.

In *preemptive* scheduling problems, the processing of some tasks can be interrupted, to be resumed at a later time. CP is generally much more successful on non-preemptive rather than preemptive scheduling problems. The standard approach to modelling interruptible tasks in constraint programming, while preserving completeness[1], is to decompose each task into as many unit-length tasks as its duration. A disjunctive resource can then be modelled as an AllDifferent constraint, a parallel-machine resource as a GlobalCardinality

[1] Without this restriction, the range of possible approaches is significantly wider.

constraint, and a cumulative resource as a general flow constraint. Another approach is to decompose the tasks into variable-size tasks. In this case, either the duration variables have a null lower bound, which severely hinders constraint propagation, or constraints have to be posted on the number of fragments and their duration, which entails searching over the different ways to split the tasks. As a result, the former approach is often the best. In either cases, the complexity of the methods heavily depends on the duration of the tasks, and therefore do not scale well in practice.

In this paper, we use the observation that when preemptive resources are disjoint, there is no need to compute the fragmentation of the tasks during search. In other words, the problem can be modelled as a constraint satisfaction problem (CSP) with two variables per task: one for its start time and one for its end time. Deciding whether a set of interruptible tasks with release and due dates can be processed on a disjunctive constraint is easy (e.g., via the *Jackson Preemptive Schedule*). Moreover, if some tentative release and due dates for the tasks pass the *overload check*[2] [29], then a feasible fragmentation of the tasks is guaranteed to exist. Therefore, provided that the disjunctive resources are *disjoint* and that constraint propagation is at least as strong as the overload check, one can simply branch on start and end time variables for every task.

This condition of disjointness is true in many problems, and there is no further restriction on the constraint graph. For instance in this paper we focus on the preemptive Job Shop Scheduling Problem (pJSSP). In this problem, the resources are naturally disjoint, but tasks requiring distinct machines can be linked by precedence constraints. Our method can be applied to several other preemptive versions of disjunctive scheduling problems (e.g., open shop scheduling where job sequences are to be decided, job shop scheduling augmented with setup times, or generalised precedences, etc.).

The paper is organised as follows: In Section 2 we recall some background and define the disjunctive preemptive constraint as well as the preemptive Job shop Scheduling problem. In Section 3 we briefly review the existing CP models for preemptive scheduling. We show that the standard approach of using the constraints AllDifferent or NoOverload on unit-length tasks are weaker than the same approach using the AllDiffPrec constraint. Then we discuss the main observations, which are not original, but are key to our main result: it is often not necessary to introduce unit-length tasks, and without unit-length tasks Edge-Finding is sufficient to enforce bounds consistency. Then we introduce a novel constraint model for the preemptive Job Shop Scheduling Problem based on these observations in Section 4 and finally we discuss the state of the art for the pJSSP and experimentally compare our approach to it in Section 5.

2 Background and Related Work

2.1 CSP and bounds consistency

A *constraint network* is a triple $(\mathbf{x}, \mathcal{D}, \mathbf{c})$ where $\mathbf{x}$ is a finite totally ordered set of *variables*, $\mathcal{D}$ is a finite set, and $\mathbf{c}$ is a finite set of *constraints*. A constraint c is a pair (S_c, P_c) where the *scope* S_c is a subset of $\mathbf{x}$ and P_c is a predicate on the variables S_c. An assignment $\sigma : \mathbf{x} \mapsto \mathcal{D}$ is a mapping from variables in $\mathbf{x}$ to values in $\mathcal{D}$, and $\sigma(x)$ denotes the value assigned to variable x by σ. Given a constraint network $(\mathbf{x}, \mathcal{D}, \mathbf{c})$, the *Constraint Satisfaction Problem (CSP)* asks whether there is an assignment σ of values in $\mathcal{D}$ to the variables $\mathbf{x}$ such that for every constraint $c \in \mathbf{c}$, the predicate P_c is true on the projection of σ onto S_c.

[2] That the total duration of any set of tasks is not larger than their execution window.

Most constraint solvers store a current domain $D(x) \subseteq \mathcal{D}$ for every variable $x \in \mathbf{x}$ and use some form of consistency reasoning to reduce these domains without losing solutions. We assume that the domain $\mathcal{D}$ is totally ordered and denote $\min(x)$ (resp. $\max(x)$) the minimum (resp. maximum) value in $D(x)$. Moreover, we denote $[l, u]$ the discrete interval containing every element in $\mathcal{D}$ that is larger than or equal to l and lower than or equal to u.

▶ **Definition 1** (Bounds Consistency). *Let c be a constraint and $x \in S_c$ be a variable constrained by c. An assignment σ is a* bound support *of the pair (x, v) in a current domain D if and only if:*

- $\sigma(x) = v$;
- *the predicate P_c is satisfied by σ (σ is said to be* consistent *and we write $P_c(\sigma) =$ `true`);*
- *and for every variable $y \in \mathbf{x} \setminus x$, $\sigma(y) \in [\min(y), \max(y)]$ (σ is said to be* valid*).*

A constraint c is bounds consistent (BC) for the current domain D if and only if, for every $x \in S_c$, $(x, \min(x))$ and $(x, \max(x))$ have a bound support in D.

A constraint network $\mathcal{N} = (\mathbf{x}, \mathcal{D}, \mathbf{c})$ is BC in a current domain D if and only if, for every $c \in \mathbf{c}$, c is BC in D.

The notion of consistency can be used to compare constraint models, and in particular compare constraints to their decompositions.

▶ **Definition 2** (Pruning power). *Let c be a constraint and $\mathcal{N} = (\mathbf{x}, \mathcal{D}, \mathbf{c})$ be a decomposition of the constraint, i.e., a constraint network such that:*

- *$S_c \subseteq \mathbf{x}$ and*
- *σ satisfies P_c if and only if, there is a solution σ' of $\mathcal{N}$ such that the projection of σ' onto S_c is equal to σ.*

We say that BC on the decomposition $\mathcal{N}$ is as strong as *BC on the constraint c if for any current domain D, it holds that $\mathcal{N}$ is BC in D implies c is BC in D. Otherwise, we say that BC on the decomposition is* weaker *than on the constraint.*

2.2 Preemptive Scheduling

The constraint PreemptiveNoOverlap ensures that a set of interruptible tasks requiring a disjunctive resource do not overlap. Let $\mathcal{T} = \{t_1, \ldots, t_n\}$ be a set of tasks, and let $\mathbf{s} = \{s_1, \ldots, s_n\}$ and $\mathbf{e} = \{e_1, \ldots, e_n\}$ be two sets of variables standing respectively for the earliest start and latest end times of the tasks, i.e., a task $t_i \in \mathcal{T}$ must be processed in the interval $[s_i, e_i)$ (closed at the start and opened at the end). Notice that although we will often use the terms "start (or end) variable" for convenience, these variables actually define an interval in which the task is processed. In particular, task t_j might not be processed at all at time s_j and may be finished at a time strictly earlier than e_j. This is apparent in Definitions 3 and 4, and the reasons for this choice are discussed in Section 3.2.1.

Moreover, let p_i be the duration of task t_i. For a set of tasks $\Omega \subseteq \mathcal{T}$, we write s_Ω for the earliest start time of any task in Ω ($s_\Omega = \min(\{s_i \mid t_i \in \Omega\})$), e_Ω for the latest end time of any task in Ω ($e_\Omega = \max(\{e_i \mid t_i \in \Omega\})$), and $p_\Omega = \sum_{t_i \in \Omega} p_i$ for the total processing time of tasks in Ω. Since the tasks are interruptible, there must exist a "fragmentation" function that maps at most one task to each time point. Let $\mathcal{H} = [0, ub)$ be a time interval where ub is some upper bound on the end times of tasks in $\mathcal{T}$, and let $\mathbf{a} = \{a_{i,\tau} \mid t_i \in \mathcal{T}, \tau \in \mathcal{H}\}$ be Boolean *activation variables*, where $a_{i,\tau}$ indicates whether task t_i is active at time τ.

▶ **Definition 3.** *PreemptiveNoOverlap (on activation variables)*

$$\textit{PreemptiveNoOverlap}(\mathcal{T}, \mathbf{a}, \mathbf{s}, \mathbf{e})$$
$$\iff$$
$$\forall \tau \in \mathcal{H} \sum_{t_i \in \mathcal{T}} a_{i,\tau} \leq 1 \wedge \forall t_i \in \mathcal{T} \sum_{\tau=s_i}^{e_i-1} a_{i,\tau} = p_i$$

In this paper we consider the case where we do not make the fragmentation function explicit (via activation variables), but rather only care about the start and end times of the tasks, while only making sure that some fragmentation function exists. This leads to Definition 4:

▶ **Definition 4.** *PreemptiveNoOverlap (on start and end variables)*

$$\textit{PreemptiveNoOverlap}(\mathcal{T}, \mathbf{s}, \mathbf{e})$$
$$\iff$$
$$\exists f : \mathcal{H} \mapsto \mathcal{T} \cup \{\emptyset\}, \forall t_i \in \mathcal{T},\ |\{\tau \in [s_i, e_i) \mid f(\tau) = t_i\}| = p_i$$

2.3 Preemptive Job Shop Scheduling (pJSSP)

In the pJSSP, a set $\mathcal{J}$ of n jobs are to be processed on a set $\mathcal{M}$ of machines. Each job $J_i \in \mathcal{J}$ consists of a sequence of n_i tasks that must be executed in order. Each task $t_{i,j} \in J_i$ must be executed on one machine $M_{i,j} \in \mathcal{M}$ with a processing time $p_{i,j} \in \mathbb{N}$. Preemption is allowed, i.e. tasks can use a machine for some time, stop to let another task be processed, and then resume at a later time. The objective is to minimise the total makespan, that is, the maximum completion time of any task (denoted $C_{\max}$).

The pJSSP is NP-hard even with two machines and three jobs ($J2|n = 3, pmtn|C_{\max}$) [7] while the non-preemptive version ($J2|n = 3|C_{\max}$) is solvable in polynomial time; an $O(r^4)$-algorithm with $r = \max_{i \in \mathcal{J}} n_i$ in given in [22]. We have observed that in most academic data sets used to benchmark job shop scheduling algorithms, there is "no recirculation", that is, jobs have exactly one task per machine. However, this particular case is also NP-hard for as few as three machines since it generalises the flow shop problem which is itself NP-hard [16]. The approach introduced in this paper applies to job shop problems with or without recirculation, although all the instances used in the experiments belong to the latter class.

3 Constraint Programming for Preemptive Scheduling

Many propagation algorithms for reasoning on resources for non-interruptible tasks rely on relaxing non-preemption, and can therefore be applied to the preemptive case with very few changes, as observed for instance in [24].

The *Edge-Finding* rule is of particular interest in the preemptive case. It relies on the *overload check* implied by the disjunctive resource constraint:

▶ **Definition 5.** *NoOverload*

$$\textit{NoOverload}(\mathcal{T}) \iff \bigwedge_{\Omega \subseteq \mathcal{T}} p_\Omega \leq e_\Omega - s_\Omega$$

In the preemptive case, the overload check is not only implied, it is equivalent to the PreemptiveNoOverlap constraint. The following theorem is a direct corollary of Proposition 3 in [8]:

▶ **Theorem 6.** $\textsc{PreemptiveNoOverlap}(\mathcal{T}, \mathbf{s}, \mathbf{e}) \iff \textsc{NoOverload}(\mathcal{T}, \mathbf{s}, \mathbf{e})$

The Edge-Finding rule consists in detecting precedence constraints whose violation would in turn make the overload check false. In the non-preemptive case, they are defined as shown in Definition 7. They are written here as constraints, but notice that they can be directly translated to a propagation rule by considering "optimistic" values for variables in the left-hand side (minima for start times and maxima for end times). Of course both constraints have a symmetric version (by reflection).

▶ **Definition 7.** *Edge-Finding (non-preemptive case)*

$$s_{\Omega\cup\{t_i\}} + p_i + p_\Omega > e_\Omega \implies s_i \geq s_\Omega + p_\Omega \qquad \forall \Omega \subseteq \mathcal{T}, \forall t_i \in \mathcal{T} \setminus \Omega$$

A slight adaptation is required for the preemptive case [4]. The precondition implies that task t_i must end last among $\Omega \cup \{t_i\}$, which is not equivalent to starting last since the task can be interrupted:

▶ **Definition 8.** *Edge-Finding (Preemptive case)*

$$s_{\Omega\cup\{t_i\}} + p_i + p_\Omega > e_\Omega \implies e_i \geq s_{\Omega\cup\{t_i\}} + p_i + p_\Omega \qquad \forall \Omega \subseteq \mathcal{T}, \forall t_i \in \mathcal{T} \setminus \Omega$$

The following theorem is a direct corollary of Proposition 9 in [4] concerning *fully elastic schedules*. A fully elastic schedule is one where tasks are preemptive and do not have a constant demand on the resource when active (however, their total *energy*, i.e., total resource demand integrated over time, is a constant). On disjunctive resource, since the demand is an integer and can only be equal to 0 (inactive) or 1 (active), fully elastic schedules and preemptive schedules are equivalent. Interestingly, this theorem shows that other rules that are useful on non-preemptive scheduling (such as the "not-first/not-last" rule) are useless in the preemptive case.

▶ **Theorem 9.** *Edge-Finding constraints are all bounds consistent on a set of tasks $\mathcal{T}$ if and only if $\textsc{PreemptiveNoOverlap}(\mathcal{T})$ is bounds consistent.*

3.1 Formulation with fragmentation

It is easy to see Definition 4 is not sufficient when the same preemptive task requires more than a single (disjunctive) resource. Consider the instance illustrated in Figure 1a. Task t_1 requires resources a and b while tasks t_2 and t_3 require resource a, and t_4 and t_5 require resource b. All start and end times are fixed, and under Definition 4, the constraint PreemptiveNoOverlap is satisfied both for the scope $(s_1, s_2, s_3, e_1, e_2, e_3)$ and $(s_1, s_4, s_5, e_1, e_4, e_5)$. However, there is no assignment of the variables $\mathbf{a} = a_{1,0}, \ldots, a_{1,5}$ which satisfies both constraints under Definition 3. In this section we review the existing formulations of the disjunctive constraint that model task fragmentation.

A standard way to model preemptive resources is to decompose each task t_i into p_i unit-length tasks, where variable $x_{i,k}$ stands for the processing time of the k-th unit of task t_i. Then a disjunctive resource can be represented as an AllDifferent constraint:

▶ **Definition 10.** *$\textsc{AllDifferent}$ decomposition of $\textsc{PreemptiveNoOverlap}$ (w.r.t. Definition 3)*

$$\textsc{PreemptiveNoOverlap}(\mathcal{T}, \mathbf{a}, \mathbf{s}, \mathbf{e}) \iff$$

$$\textsc{AllDifferent}(\{x_{i,k} \mid t_i \in \mathcal{T}\ \forall k \in [0, p_i)\}) \quad (1)$$

$$\forall t_i \in \mathcal{T}\ \forall k \in [0, p_i)\ \forall \tau \in \mathcal{H} \qquad x_{i,k} = \tau \iff a_{i,\tau} \quad (2)$$

$$\forall t_i \in \mathcal{T}\ \forall k \in [0, p_i) \qquad s_i \leq x_{i,k} < e_i \quad (3)$$

	p_i	s_i	e_i	Res.
t_1	3	0	6	a and b
t_2	1	1	2	a
t_3	1	3	4	a
t_4	1	2	3	b
t_5	1	4	5	b

(a) with vs. without fragmentation

	p_i	s_i		e_i	
t_1	2	0	3	2	5
t_2	2	0	3	2	5
t_3	2	0	5	2	7

(b) AllDifferent decomposition

Figure 1 An example showing that activation variables are necessary when a task requires several distinct resources (Figure 1a), and an example showing that the decomposition using the AllDifferent constraint hinders propagation (Figure 1b).

A similar decomposition holds for the constraint PreemptiveNoOverlap$(\mathbf{s}, \mathbf{e})$ when we ignore the activation variables, and hence Constraint (2).

These two decompositions hinder propagation, i.e., BC on the decomposition is not equivalent to BC on the global constraint.

▶ **Theorem 11.** *Bounds consistency on Constraints 1, 2 and 3 is weaker than bounds consistency on* PreemptiveNoOverlap$(\mathcal{T}, \mathbf{s}, \mathbf{e})$.

Proof. Consider the three tasks shown in Figure 1b. The AllDifferent constraint has 6 variables in both decompositions, and there is no Hall interval and hence is BC. Therefore, the channelling constraints are also BC. Yet the values 2 to 5 are not BC for e_3 in the global constraint, since that would produce an overload in the interval $[0, 5)$. ◀

The decomposition is weaker because the p_i units of task t_i are interchangeable. Indeed we may add the following symmetry breaking constraints:

$$\forall t_i \in \mathcal{T}\ \forall k \in [1, p_i) \qquad x_{i,k-1} < x_{i,k} \tag{4}$$

However, there is a quadratic algorithm to achieve BC on the conjunction of an AllDifferent constraint and a set of binary precedence constraints (the AllDiffPrec constraint [5]), and which therefore be used to achieve BC on the conjunction of Constraints 1 and 4. Let this formulation be "the AllDiffPrec decomposition".

▶ **Theorem 12.** *Bounds consistency on the* AllDiffPrec *decomposition is as strong as bounds consistency on* PreemptiveNoOverlap$(\mathcal{T}, \mathbf{s}, \mathbf{e})$.

Proof. Let $\mathcal{T}$ be a set of tasks with start and end variables $\mathbf{s}$, $\mathbf{e}$, and suppose that PreemptiveNoOverlap$(\mathcal{T}, \mathbf{s}, \mathbf{e})$ is not BC. Then there exists a task $t_i \in \mathcal{T}$ whose upper bound is not BC. By Theorem 6, it means that there exist $v \leq \max(e_i)$ and $\Omega \subset \mathcal{T}$ such that $p_\Omega + p_i > \max(e_\Omega, v) - \min(s_\Omega, \min(s_i))$. Now, consider the assignment $x_{i,p_i} \leftarrow v$ in the decomposition. In order to satisfy the precedences $x_{i,k-1} < x_{i,k}$ for $k \in [1, p_i)$, all these variables must take values less than or equal to v. Therefore, in the decomposition, there are $p_\Omega + p_i$ variables which must take a value in the interval $[\min(s_\Omega, \min(s_i)), \max(e_\Omega, v))$ which is unfeasible. It follows that the assignment $x_{i,p_i} \leftarrow v$ is not BC for the AllDiffPrec constraint. ◀

This decomposition, however, requires $O(N)$ variables with $N = \sum_{t_i \in \mathcal{T}} p_i$, and BC can be achieved in $O(N^2)$ time. However, it can be achieved in a time complexity that does not depend on N by direct application of Theorem 9. Indeed, there are algorithms

in $O(n \log n)$ [29] or even in linear time (after sorting) [13] to achieve BC on at least one Edge-Finding constraint. Moreover, since there are at most n^2 precedences to enforce, achieving BC on the PreemptiveNoOverlap constraint can be done in time polynomial in n only.[3]

3.2 Formulation without fragmentation

When resources are *disjoint*, that is, when no task requires more than a single resource, then the problem can be modelled without representing task fragmentation. Indeed, Definition 4 ensures that a fragmentation such that no two tasks requiring that resource are processed simultaneously, and each task t_i is processed within the time interval $[s_i, e_i)$, in other words, we have:

$$\exists \mathbf{a}\ \textsc{PreemptiveNoOverlap}(\mathcal{T}, \mathbf{a}, \mathbf{s}, \mathbf{e}) \iff \textsc{PreemptiveNoOverlap}(\mathcal{T}, \mathbf{s}, \mathbf{e})$$

Therefore, when activation variables ($\mathbf{a}$) are not constrained otherwise, the two formulations are equivalent.

Checking this constraint can be done efficiently: the Jackson Preemptive Schedule algorithm [9, 18] finds a fragmentation, or proves that none exists in $O(n \log n)$ time. Moreover, Theorem 6 entails that one does not need to find a witness fragmentation if the constraint propagation of the disjunctive constraint involves the overload check.

3.2.1 Monotonicity

Notice that the definition of the PreemptiveNoOverlap constraint does not force a task t_i to be in process at time s_i, nor at time $e_i - 1$. In other words, start and end times are simply bounds within which the task can be processed. It follows that this constraint is *monotonic*: decreasing the start time of a task (or increasing its end time) in a satisfying assignment can never make this assignment inconsistent.

▶ **Definition 13** (Monotonic constraints). *Let $\sigma_{x \leftarrow v}$ be the assignment that associates value v to variable x and that is equal to σ otherwise. We say that a constraint c is monotonic with respect to a function $f : \mathbf{x} \times \mathcal{D} \mapsto \mathcal{D}$ if and only if:*

$$P_c(\sigma) = \texttt{true} \implies (\forall x \in S_c,\ P_c(\sigma_{x \leftarrow f(x, \sigma(x))}) = \texttt{true})$$

▶ **Lemma 14.** *The constraint $\textsc{PreemptiveNoOverlap}(\mathcal{T}, \mathbf{s}, \mathbf{e})$ is monotonic with respect to any function that is non-increasing for start-time variables, or non-decreasing for end-time variables.*

Proof. The fragmentation of a task remains valid if its start time is decreased or its end time is increased. ◀

▶ **Corollary 15.** *If the constraint $\textsc{PreemptiveNoOverlap}(\mathcal{T}, \mathbf{s}, \mathbf{e})$ is satisfiable, then, for any $t_i \in \mathcal{T}$, the assignments $s_i \leftarrow \min(s_i)$ and $e_i \leftarrow \max(e_i)$ are bounds consistent.*

Proof. If the constraint is satisfiable, there exists a consistent and valid assignment, and by Lemma 14, changing the value of s_i to $\min(s_i)$ (resp. e_i to $\max(e_i)$) is a non-increasing (resp. non-decreasing) operation and hence yields a consistent and valid assignment. ◀

[3] In practice a fix-point can be reached in far fewer iterations.

There are two consequences to the PreemptiveNoOverlap constraint being monotonic.

Firstly, achieving bounds consistency on the PreemptiveNoOverlap constraint can only prune the upper (resp. lower) bound of the start (resp. end) time variables. However, bounds of end time variables could be pruned beyond BC without losing solutions. For instance, assume that task t_i is such that $s_i = 0, e_i = 3$ and $p_i = 3$. Clearly, this task requires the resource on the whole interval $[0, 3)$ and therefore no other task can start at a time point earlier than 3. This corresponds to achieving BC on a restriction of Definition 4 where we constrain every task $t_j \in \mathcal{T}$ to be in process at time s_j and at time $e_j - 1$. We have experimented with this formulation, and BC can be achieved in the same time complexity as enforcing Edge-Finding, using a slight generalisation of the propagation algorithm for BC on the AllDifferent constraint [27]. However, besides complexifying the definitions and the algorithm, it turns out that achieving this extra pruning is counter-productive, at least on pJSSP benchmarks.

Secondly, in the job shop scheduling problem, the only constraints besides disjunctive resources are the chain of precedences to represent the job sequences. Therefore, the start time of a task can always be extended to the end time of the previous task on that job (or to 0 if it is the first task) without invalidating the schedule. Similarly, its end time can be extended to the start time of the next task in that job. As a result, we can assume that a task ends exactly when the next task of its job starts and forbid any idle gap between the tasks of a job.

4 A Constraint Programming Approach to pJSSP

From the observation made in Section 3.2, we can propose the following constraint model for the preemptive job shop scheduling problem, with $s_{i,j}$ (resp. $e_{i,j}$) standing for the start (resp. end) variable associated to the j-th task of job i, and with $\mathcal{T} = \{t_{i,j} \mid J_i \in \mathcal{J}, \forall j \in [1, n_i)\}$.

$$\min C_{\max} \tag{5}$$

$$s.t. \quad C_{\max} \geq e_{i,n_i} \qquad \forall J_i \in \mathcal{J} \tag{6}$$

$$e_{i,j} \geq s_{i,j} + p_{i,j} \qquad \forall J_i \in \mathcal{J},\ \forall j \in [1, n_i] \tag{7}$$

$$e_{i,j} \leq s_{i,j+1} \qquad \forall J_i \in \mathcal{J},\ \forall j \in [1, n_i) \tag{8}$$

$$\textsc{PreemptiveNoOverlap}(\mathcal{T}_m, \mathbf{s}_m, \mathbf{e}_m) \qquad \forall m \in \mathcal{M} \tag{9}$$

The objective variable $C_{\max}$ represents the *makespan*, that is, the maximum completion time of any task (Constraint 6). Constraints 7 and 8 encode respectively the durations of the task, and the job sequences. As discussed in this section, Constraints 9 (with $\mathcal{T}_m = \{t_{i,j} \mid M_{i,j} = m, J_i \in \mathcal{J}\}$, $\mathbf{s}_m = \{s_{i,j} \mid t_{i,j} \in \mathcal{T}_m\}$ and $\mathbf{e}_m = \{e_{i,j} \mid\mid t_{i,j} \in \mathcal{T}_m\}$) are sufficient to ensure that a preemptive schedule exists, and can be computed efficiently once all start and end time variables are fixed.

Moreover, because of Corollary 15, we know that extending the end time of a task cannot violate the resource constraints. Since the only other constraints are chains of precedences, extending the end time of task up to the start time of the next task in its job (or extending its start time to the end time of the preceding task in the job) cannot violate any constraint.

We can therefore replace Constraints 6 and 7 with the following constraints:

$$s_{i,0} = 0 \qquad \forall i \in \mathcal{J} \tag{10}$$

$$e_{i,j} = s_{i,j+1} \qquad \forall i \in \mathcal{J},\ \forall j \in [1, n_i) \tag{11}$$

$$e_{i,n_i} = C_{\max} \qquad \forall i \in \mathcal{J} \tag{12}$$

Constraints 10 and 12 force the start (resp. end) time of the first (resp. last) task of every job to be equal to 0 (resp. the makespan), and Constraint 11 ensures that there is no gap between the end of a task and the start of the next task on its job. With these constraints, dominated solutions that leave a gap between two consecutive tasks of the same job are pruned out.

5 Experimental Results

In this section we compare our approach with the state-of-the-art approaches for the pJSSP. We first describe the two comparison methods: a recent dedicated branch-and-bound algorithm and the commercial solver CP Optimizer.

5.1 State-of-the-art Approaches

Most solution methods for the pJSSP are approximation algorithms [3, 15, 20] and heuristics [31]. Among the exact methods, Dantzig [10] introduced a linear programming model based on time index. Le Pape and Baptiste [24, 25] proposed a branch-and-bound procedure using classical constraint propagation techniques (timetable, disjunctive constraints and Edge-Finding) extended to preemptive problems. Ebadi and Moslehi have recently proposed two exact solution methods for the pJSSP, a mixed-integer programming (MIP) approach [11], and a dedicated branch-and-bound algorithm [12]. As our method, the MIP model requires no activation variable. It only involves variables for the start and times of the tasks, with the same guarantee that feasible (resp. optimal) solutions of this MIP correspond to feasible (resp. optimal) complete schedules, which task fragmentation can be computed e.g., via application of the Jackson's preemptive algorithm. However, in order to guarantee that the overload check is satisfied for a given resource, the model involves a set of linear constraints of size exponential in the number of tasks requiring this resource. This MIP model is less efficient than the dedicated branch-and-bound method proposed by the same authors, and hence we used the latter as reference in our experimental evaluation.

We do not compare with the recent method introduced in [21] in our experiments since this approach deals with the more general preemptive *and flexible JSSP*. It uses a logic-based Benders decomposition that splits the problem into a master problem of assigning operations to machines and into non-flexible pJSSP subproblems. The master problem is solved by mixed-integer programming while the subproblems are solved by existing approaches, such as the one introduced in this paper.

5.1.1 Dedicated Branch-and-Bound

Ebadi and Moslehi's branch-and-bound procedure [12] employs a depth-first search strategy to explore the set of feasible schedules without *proactively* creating unit-length tasks.

However, since the propagation in their method is not as strong as the overload check, a different technique is used to ensure that the produced schedule follows the Jackson's preemptive rule on each machine: unit-length tasks are created lazily when branching. In the search tree, each node represents a partial schedule with a set of already scheduled unit-length tasks and a disjunctive graph representing the current precedence relations. At the root node, the set of scheduled tasks is empty, and the arcs of the graph are only the precedences between the tasks of the same job. At each decision point, the machine processing the non-scheduled unit-length task with the smallest availability date is selected. The branching strategy consists in creating a node for each such task on this machine, with

this task scheduled at its earliest possible start time. Moreover, dominance rules ensure that many unit-length tasks can be created and added to the partial schedule at once as edges in the disjunctive graph. Lower bounds are computed at each node, based on the disjunctive graph, for pruning the search tree.

This method improved the state of the art for this problem at time of publication, and in particular Le Pape and Baptiste's CP model. It was the first to solve large instances (up to 50 jobs and 10 machines) to optimality. To our knowledge, this is currently the most efficient method to solve the pJSSP problem.

5.1.2 CP Optimizer model

IBM CP Optimizer solver is the most efficient off-the-shelf tool in many scheduling problems. We describe in this section the standard model for the preemptive job shop scheduling problem in CP Optimizer.

CP Optimizer allows the use of specific decision variables and constraints. In particular, interval variables can be used to represent the time during which a task is processed. Interval variables are defined by a start value, an end value and a size, which are the decision variables of the problem. We denote $t_{i,j}$ this interval variable, whose start and end time variables correspond to s_i and e_i respectively. Moreover, in this model, each preemptive task is divided into unit-length parts. Therefore, for each task $t_{i,j}$, we introduce $p_{i,j}$ unit-length interval variables $x_{i,j,k}$ with $k \in [1, p_{i,j}]$ besides the interval variable $t_{i,j}$.

The problem is described as follows:

$$\min C_{\max} \tag{13}$$

$$s.t. \quad C_{\max} \geq e_{i,n_i} \qquad \forall J_i \in \mathcal{J} \tag{14}$$

$$EndBeforeStart(t_{i,j}, t_{i,j+1}) \qquad \forall J_i \in \mathcal{J},\ \forall j \in [1, n_i) \tag{15}$$

$$EndBeforeStart(x_{i,j,k}, x_{i,j,k+1}) \qquad \forall J_i \in \mathcal{J},\ \forall j \in [1, n_i],\ \forall k \in [1, p_{i,j}) \tag{16}$$

$$Span(t_{i,j}, x_{i,j,k} : \forall k \in [1, p_{i,j}]) \qquad \forall J_i \in \mathcal{J},\ \forall j \in [1, n_i] \tag{17}$$

$$\textsc{NoOverlap}(x_{i,j,k} : \forall J_i \in \mathcal{J},\ \forall j \in [1, n_i], \forall k \in [1, p_{i,j}] \mid M_{i,j} = m) \qquad \forall m \in \mathcal{M} \tag{18}$$

The objective function (13) is to minimise the makespan. Constraints (14) define the makespan. The global constraint *EndBeforeStart* is used to model the precedence constraints, as in the two following constraint sets. Constraints (15) ensure that the tasks of the same job will be processed with respect to the job sequence. Constraints (16) aim at ordering the parts of the task and so avoid symmetries, and ensure that each part is treated one after the other. With the *Span* global constraint, Constraints (17) are used to ensure that task interval spans over all its processing parts (i.e., each task starts with its first part and ends with its last part). With the *NoOverlap* global constraint, Constraints (18) forbid the overlapping of tasks processed on the same machine. We denote this model CPO$^{p=1}$, and we experimented with several variants of this models where a task t_j is cut in fewer than p_j pieces. These variants are sound but incomplete: the optimal schedule on these models is feasible but not necessarily optimal for the original problem. However, the idea is that they should be easier to solve and hopefully approximate the optimal solution.

- CPO$^{p=\ell}$ refers to the model where each task $t_{i,j}$ is cut into $\lfloor \frac{p_{i,j}}{\ell} \rfloor$ subtasks of duration ℓ and one task of duration $p_{i,j} \bmod \ell$.
- CPO$^{n=\ell}$ refers to the model where each task $t_{i,j}$ is cut into ℓ tasks of variable duration but whose total is constrained to be $p_{i,j}$ (Constraint 19).

$$\sum_{k=1}^{\ell} sizeOf(x_{i,j,k}) = p_{i,j} \quad \forall J_i \in \mathcal{J},\ \forall j \in [1, n_i] \tag{19}$$

To avoid symmetries, we made sub-task interval variables optional and add the following constraints:

$$PresenceOf(x_{i,j,k+1}) \implies PresenceOf(x_{i,j,k}) \quad \forall J_i \in \mathcal{J},\ \forall j \in [1, n_i],\ \forall k \in [1, \ell) \tag{20}$$

$$EndOf(x_{i,j,k}) < StartOf(x_{i,j,k+1}) \quad \forall J_i \in \mathcal{J},\ \forall j \in [1, n_i],\ \forall k \in [1, \ell) \tag{21}$$

Constraints (20) ensure that a sub-task can only be present if the previous sub-task is also present. Constraints (21) guarantees that two successive pieces of the same task do not immediately follow each other (a task is split only if it has preemption).

Interestingly, the CP Optimizer model using the constraint AllDifferent (as shown in Definition 10, with the symmetry breaking Constraints 4) instead of NoOverlap turned out to be much less efficient in our experiments.[4] This is surprising because the latter constraint is more general and yet equivalent when tasks are unit-length. We conjecture that this could be explained by some hidden preprocessing in CP Optimizer.

5.2 Experimental protocol

We used some standard benchmark instances available in the literature [1, 2, 14, 23, 28]. These instances were proposed for the JSSP without preemption, but are often used in the preemptive case as well. Characteristics of these benchmarks are summarised in Table 1. For each benchmark, we report the number of instances (63 in total) that compose it, the size of these instances (number of jobs $\times$ number of machines) as well as the intervals the processing times are generated from. Detailed information on these instances is presented in [19].

All experiments were performed on three cluster nodes with Intel Xeon E5-2695 v4 CPU at 2.1 GHz with a 1 hour time cutoff. The branch-and-bound algorithm is implemented in C++, the exact CPO$^{p=1}$ model and all of its variants (CPO$^{p=\ell}$ and CPO$^{n=\ell}$ for $\ell \in \{3, 10\}$) are implemented with the C++ interface of CP Optimizer 12.10.

Our approach was implemented in C++ using Mistral [17][5] with the following search strategy (corresponding to the default settings): the variable ordering uses the *minimal ratio between domain size and weighted degree* heuristic [6], with an exponential decay on the weights of 0.96 and with the *last conflict* procedure [26]. The value ordering uses binary branching with the constraints $x \leq \lfloor \min(x) + \max(x) \rfloor / 2$ and $x > \lfloor \min(x) + \max(x) \rfloor / 2$, and a geometric restart policy [30] starting at 200 fails and increasing by a factor 1.05.

5.3 Numerical results

Figure 2a shows how many instances are optimally solved by each method as a function of time. We include the results of the incomplete variants (dashed lines) although these proofs are weaker: they show that there is no better solution for the restricted model. Nonetheless, CPO$^{p=\ell}$ obtains fewer such proofs for $\ell < 10$, and CPO$^{n=\ell}$ can only prove a single instance.

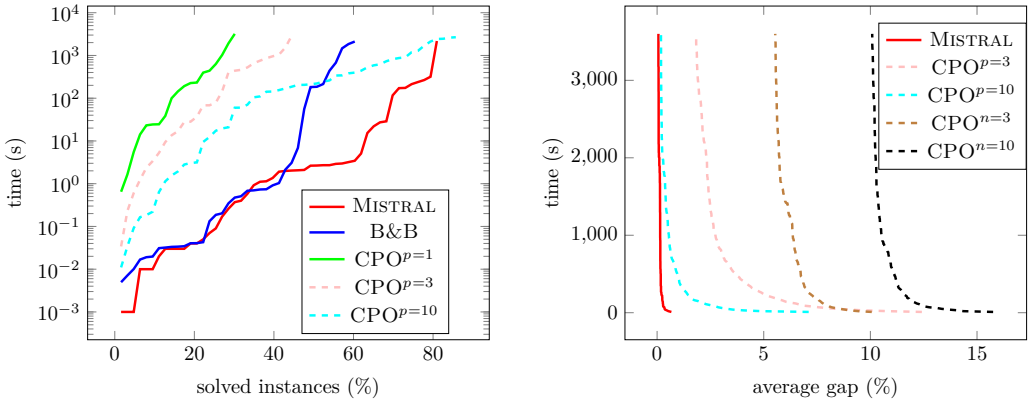
[4] Hence we only report results for the best model using NoOverlap.

[5] The source code of Mistral is available here: `https://github.com/ehebrard/Mistral-2.0` and features the model used in our experiments.

Table 1 JSSP instances characteristics.

Data set	Reference	Number of instances	Sizes	Processing times
`ft6,10,20`	[14]	3	6× 6, 10× 10, 20× 20	[1,10],[1,99]
`la01-40`	[23]	40	10× 5, 15× 5, 20× 5, 10× 10, 15× 10, 20× 10, 30× 10, 15× 15	[5,99]
`abz5-9`	[1]	5	10× 10, 20× 15	[50,100], [25,100], [11,40]
`orb1-10`	[2]	10	10× 10	[5,99]
`swv16-20`	[28]	5	50× 10	[1,100]

Essentially, among exact methods, CP Optimizer (CPO$^{p=1}$) solves far fewer instances than other methods in the time available and is slower on the instances it does solve. For the easiest 50% of instances, both the branch-and-bound method (B&B) and Mistral can solve them quickly, in about 10 seconds. For the other instances, Mistral is faster and manages to solve 80% of the instances to optimality against 60% for the branch-and-bound.



(a) Number of solved instances over time. **(b)** Gap over time.

Figure 2 Number of proofs and gap to the best overall solution over time. Dashed lines correspond to incomplete methods.

Figure 2b shows the average gap to the best known solution over time for Mistral and four approximate methods namely CPO$^{p=3}$, CPO$^{p=10}$, CPO$^{n=3}$ and CPO$^{n=10}$. We observe that none of these variants can find better solutions than Mistral, even considering a short calculation time. We also notice that the variants that considers fixed subtasks duration (CPO$^{p=\ell}$) are more efficient than the variant that considers a fixed number of subtasks (CPO$^{n=\ell}$) and that for these two variants, the fewer the subtasks, the more efficient is the method. In fact CPO$^{p=10}$ finds better solutions than Mistral on two instances: `abz7` and `abz8`.

Results on individual instances are reported in Table 2. Our approach is better than the branch-and-bound on all but one instance: `orb01` where the latter method proves optimality in 2118 seconds whereas Mistral does not return a proof within one hour. Moreover, within the one hour cutoff, it finds the best solution over all of the methods considered in these experiments on all but two of the instances, for which approximate models are more efficient.

Table 2 Results on every benchmark instance, proven optimal schedules are marked with a "*", best $C_{\max}$ are in **bold font**.

Inst.	MISTRAL		B&B		CPO$^{p=1}$		CPO$^{p=10}$		CPO$^{n=3}$	
	$C_{\max}$	time (s)	$C_{\max}$	time (s)	$C_{\max}$	time (s)	$C_{\max}$	time (s)	$C_{\max}$	time (s)
abz5	**1203***	213.22	1212	3600.00	1299	3600.00	1204	817.51	1266	3600.00
abz6	**924***	27.28	**924***	185.14	961	3600.00	**924**	205.52	957	3600.00
abz7	681	3600.00	749	3600.00	723	3600.00	**672**	3600.00	746	3600.00
abz8	694	3600.00	750	3600.00	723	3600.00	**677**	3600.00	766	3600.00
abz9	**691**	3600.00	752	3600.00	751	3600.00	695	3600.00	817	3600.00
ft06	**54***	< 0.01	**54***	< 0.01	**54***	0.65	55	0.02	**54**	3600.00
ft10	**900***	238.81	**900***	1846.53	955	3600.00	**900**	526.71	1035	3600.00
ft20	**1165***	0.93	**1165***	6.89	1207	3600.00	**1165**	260.83	1204	3600.00
la01	**666***	< 0.01	**666***	0.02	**666***	24.59	**666**	0.17	670	3600.00
la02	**655***	0.03	**655***	0.05	**655***	631.14	**655**	20.91	692	3600.00
la03	**597***	0.08	**597***	0.04	**597**	3600.00	**597**	19.64	635	3600.00
la04	**567***	0.06	**567***	0.14	583	3600.00	**567**	17.35	599	3600.00
la05	**593***	< 0.01	**593***	< 0.01	**593***	1.65	**593**	0.04	**593**	3600.00
la06	**926***	< 0.01	**926***	0.02	**926***	5.52	**926**	0.1	**926**	3600.00
la07	**890***	0.03	**890***	0.05	**890***	38.49	**890**	3.05	**890**	3600.00
la08	**863***	0.04	**863***	0.04	**863***	144.41	**863**	2.81	**863**	3600.00
la09	**951***	0.04	**951***	0.02	**951***	23.18	**951**	0.64	**951**	3600.00
la10	**958***	< 0.01	**958***	0.02	**958***	24.34	**958**	0.19	**958**	3600.00
la11	**1222***	0.03	**1222***	0.04	**1222***	1025.62	**1222**	3.15	**1222**	3600.00
la12	**1039***	0.03	**1039***	0.05	**1039***	396.78	**1039**	1.14	**1039**	3600.00
la13	**1150***	< 0.01	**1150***	0.04	**1150***	99.89	**1150**	1.56	**1150**	3600.00
la14	**1292***	0.02	**1292***	0.04	**1292***	14.17	**1292**	0.23	**1292**	3600.00
la15	**1207***	0.18	**1207***	0.19	**1207***	1920.59	**1207**	12.38	**1207**	3600.00
la16	**934***	22.03	**934**	3600.00	961	3600.00	**934**	244.25	997	3600.00
la17	**747***	0.1	759	3600.00	794	3600.00	**747**	110.23	793	3600.00
la18	**822***	2.65	**822***	676.12	850	3600.00	**822**	185.64	864	3600.00
la19	**812***	318.24	**812***	1469.22	825	3600.00	814	902.75	894	3600.00
la20	**871***	3.21	892	3600.00	922	3600.00	875	342.32	926	3600.00
la21	**1033***	2179	1110	3600.00	1121	3600.00	**1033**	2674.68	1122	3600.00
la22	**913***	2.92	930	3600.00	982	3600.00	**913**	1695.98	1005	3600.00
la23	**1032***	0.38	**1032***	1.04	1054	3600.00	**1032**	221.06	1039	3600.00
la24	**909**	3600.00	939	3600.00	973	3600.00	910	3600.00	1001	3600.00
la25	**947**	3600.00	983	3600.00	1071	3600.00	**947**	2428.74	1073	3600.00
la26	**1218***	3.45	1232	3600.00	1386	3600.00	**1218**	733	1272	3600.00
la27	**1235***	116.59	1346	3600.00	1360	3600.00	**1235**	2458.09	1337	3600.00
la28	**1216***	2.72	1255	3600.00	1402	3600.00	**1216**	1282.21	1299	3600.00
la29	**1173**	3600.00	1225	3600.00	1325	3600.00	1196	3600.00	1283	3600.00
la30	**1355***	0.58	**1355***	0.51	1499	3600.00	**1355**	143.42	1396	3600.00
la31	**1784***	1.91	**1784***	2.13	1835	3600.00	**1784**	60.13	1790	3600.00
la32	**1850***	1.12	**1850***	0.21	1874	3600.00	**1850**	104.93	1850	3600.00
la33	**1719***	2.06	**1719***	0.35	1817	3600.00	**1719**	59.96	1719	3600.00
la34	**1721***	2.1	**1721***	0.92	1836	3600.00	**1721**	397.13	1768	3600.00

Table 2: continued from previous page

Inst.	MISTRAL		B&B		CPO$^{p=1}$		CPO$^{p=10}$		CPO$^{n=3}$	
	$C_{\max}$	time (s)	$C_{\max}$	time (s)	$C_{\max}$	time (s)	$C_{\max}$	time (s)	$C_{\max}$	time (s)
la35	**1888***	1.37	**1888***	0.48	1944	3600.00	**1888**	587.35	1894	3600.00
la36	**1252**	3600.00	1297	3600.00	1358	3600.00	**1252**	3600.00	1360	3600.00
la37	**1397***	2.72	1411	3600.00	1500	3600.00	**1397**	2609.27	1503	3600.00
la38	**1175**	3600.00	1255	3600.00	1308	3600.00	1191	3600.00	1350	3600.00
la39	**1221***	2.98	1362	3600.00	1389	3600.00	1221	2119.53	1378	3600.00
la40	**1199**	3600.00	1311	3600.00	1357	3600.00	1234	3600.00	1332	3600.00
orb01	**1035**	3600.00	**1035***	2118.7	1115	3600.00	1036	3600.00	1114	3600.00
orb02	**864***	174.17	869	3600.00	887	3600.00	865	621.22	943	3600.00
orb03	**973**	3600.00	1054	3600.00	1043	3600.00	975	1191.62	1093	3600.00
orb04	**980***	266.45	**980***	182.2	1023	3600.00	981	382.11	1045	3600.00
orb05	**849***	28.81	852	3600.00	870	3600.00	853	339.27	977	3600.00
orb06	**985**	3600.00	997	3600.00	1109	3600.00	**985**	870.98	1079	3600.00
orb07	**389***	355.36	**389***	439.51	395	3600.45	390	127.81	**389**	1511.23
orb08	**894***	0.26	**894***	55.94	959	3600.00	**894**	169.13	960	3600.00
orb09	**917***	2.67	**917***	214.58	969	3600.00	920	154.89	988	3600.00
orb10	**930***	15.36	941	3600.00	1011	3600.00	**930**	211.73	986	3600.00
swv16	**2924***	2.03	**2924***	0.69	**2924***	191.93	**2924**	1.94	**2924**	3600.00
swv17	**2794***	0.41	**2794***	0.7	**2794***	3203.24	**2794**	9.5	**2794**	3600.00
swv18	**2852***	2	**2852***	0.74	**2852***	232.57	**2852**	70.48	**2852**	3600.00
swv19	**2843***	5.08	**2843***	3.09	2970	3600.00	**2843**	141.85	**2843**	3600.00
swv20	**2823***	1.14	**2823***	0.75	**2823***	226.14	**2823**	204.49	**2823**	3600.00

5.4 Evaluation of the compact model

Finally, we conducted further experiments to assess the gain attributable to the addition of Constraints 10, 11 and 12 in order to reduce the number of variables and eliminate solutions that leave a gap between two consecutive tasks of the same job. We ran MISTRAL on the basic model (i.e., without Constraints 10, 11 and 12) on the same benchmark instances. We only present aggregated data here.

The conclusion of these experiments is that both models (with or without) those constraints are fairly equivalent when considering the objective value only. The average gain, on the data set I containing only instances that are not proven optimal by both models, is:

$$\frac{1}{|I|}\sum_{i\in I}\frac{C_{\max}(i)-C^*_{\max}(i)}{C_{\max}(i)} = 0.03$$

where $C^*_{\max}(i)$ denote the objective value for instance i *with* the extra constraints and $C_{\max}(i)$ the objective value *without* these constraints. The difference is extremely small, and either model can be best on a given instance.

On instances that were proven optimal, however, the difference is clear and significant: proving optimality is done in 32.76 seconds on average with the extra constraints, whereas it takes 81.42 seconds on average without them. Moreover, one instance (la21) can only be proven optimal within the 1h time cutoff when the extra constraints are used.

6 Conclusion

In this paper, we introduced a CP model for the preemptive job shop scheduling problem, and our experimental evaluation shows that it significantly improves the state of the art for this problem. The key aspect of this approach is the observation that when resources are disjoint, the Edge-Finding propagation algorithm guarantees that a preemptive schedule exists, without the need to explicitly compute a fragmentation of the tasks. This approach generalises to all disjunctive scheduling problems where resources are disjoint.

Extending this approach to general resource hypergraphs is an interesting avenue for future work. It could for instance be done in a decomposition scheme whereby after solving the model described in this paper, unit-length tasks are added, however only for those tasks whose fragmentations on two resources are in conflict.

References

1 Joseph Adams, Egon Balas, and Daniel Zawack. The Shifting Bottleneck Procedure for Job shop Scheduling. *Management science*, 34(3):391–401, 1988.

2 David Applegate and William Cook. A Computational Study of the Job-shop Scheduling Problem. *ORSA Journal on computing*, 3(2):149–156, 1991.

3 Nikhil Bansal, Tracy Kimbrel, and Maxim Sviridenko. Job Shop Scheduling with Unit Processing Times. *Mathematics of Operations Research*, 31(2):381–389, 2006.

4 Philippe Baptiste, Claude Pape, and Wim Nuijten. *Constraint-Based Scheduling*. Kluwer Academic Publishers, 2001.

5 Christian Bessiere, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh. The AllDifferent Constraint with Precedences. In *Proceedings of CPAIOR 2011*, pages 36–52, 2011.

6 Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting Systematic Search by Weighting Constraints. In Ramón López de Mántaras and Lorenza Saitta, editors, *Proceedings of ECAI 2004*, pages 146–150, 2004.

7 Peter Brucker, Svetlana A. Kravchenko, and Yuri N. Sotskov. Preemptive Job-shop Scheduling Problems with a Fixed Number of Jobs. *Mathematical Methods of Operations Research*, 49(1):41–76, 1999.

8 Jacques Carlier. The One-machine Sequencing Problem. *European Journal of Operational Research*, 11(1):42–47, 1982.

9 Jacques Carlier and Eric Pinson. A Practical Use of Jackson's Preemptive Schedule for Solving the Job-Shop Problem. *Annals of Operations Research*, 26:269–287, 1990.

10 George B. Dantzig. A Machine-job Scheduling Model. *Management Science*, 6(2):191–196, 1960.

11 Abbas Ebadi and Ghasem Moslehi. Mathematical Models for Preemptive Shop Scheduling Problems. *Computers & Operations Research*, 39(7):1605–1614, 2012.

12 Abbas Ebadi and Ghasem Moslehi. An Optimal Method for the Preemptive Job Shop Scheduling Problem. *Computers & Operations Research*, 40(5):1314–1327, 2013.

13 Hamed Fahimi and Claude-Guy Quimper. Linear-Time Filtering Algorithms for the Disjunctive Constraint. In *Proceedings of AAAI'2014*, pages 2637–2643, 2014.

14 Henry Fisher and G.L. Thompson. Probabilistic Learning Combinations of Local Job-shop Scheduling Rules. In J.F. Muth and G.L. Thompson, editors, *Industrial scheduling*, pages 225–251. Prentice-Hall, Englewood Cliffs, N.J., 1963.

15 Leslie Ann Goldberg, Mike Paterson, Aravind Srinivasan, and Elizabeth Sweedyk. Better Approximation Guarantees for Job-shop Scheduling. *SIAM Journal on Discrete Mathematics*, 14(1):67–92, 2001.

16 Teofilo Gonzalez and Sartaj Sahni. Flowshop and Jobshop Schedules: Complexity and Approximation. *Operations research*, 26(1):36–52, 1978.

17 Emmanuel Hebrard. Mistral, a Constraint Satisfaction Library. *Third International CSP Solver Competition*, pages 31–39, 2008. URL: `http://www.cril.univ-artois.fr/CPAI08/Competition-08.pdf`.

18 W.A. Horn. Some Simple Scheduling Algorithms. *Naval Research Logistics Quarterly*, 21(1):177–185, 1974.

19 Anant Singh Jain and Sheik Meeran. Deterministic Job-shop Scheduling: Past, Present and Future. *European journal of operational research*, 113(2):390–434, 1999.

20 Klaus Jansen, Roberto Solis-Oba, and Maxim Sviridenko. Makespan Minimization in Job Shops: A Linear Time Approximation Scheme. *SIAM Journal on Discrete Mathematics*, 16(2):288–300, 2003.

21 Carla Juvin, Laurent Houssin, and Pierre Lopez. Logic-based Benders decomposition for the preemptive flexible job-shop scheduling problem. *Comput. Oper. Res.*, 152:106156, 2023. `doi:10.1016/j.cor.2023.106156`.

22 Svetlana A. Kravchenko and Yuri N. Sotskov. Complexity of the Two Machine Job-shop Scheduling Problem with a Fixed Number of Jobs. *Central European Journal for Operations Research and Economics*, 1995.

23 Stephen Lawrence. Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement). *Graduate School of Industrial Administration, Carnegie-Mellon University*, 1984.

24 Claude Le Pape and Philippe Baptiste. Resource Constraints for Preemptive Job-shop Scheduling. *Constraints*, 3:263–287, 1998.

25 Claude Le Pape and Philippe Baptiste. Heuristic Control of a Constraint-based Algorithm for the Preemptive Job-shop Scheduling Problem. *Journal of Heuristics*, 5:305–325, 1999.

26 Christophe Lecoutre, Lakhdar Sais, Sébastien Tabary, and Vincent Vidal. Last Conflict Based Reasoning. In *Proceedings of ECAI 2006*, pages 133–137, 2006.

27 Alejandro López-Ortiz, Claude-Guy Quimper, John Tromp, and Peter van Beek. A Fast and Simple Algorithm for Bounds Consistency of the AllDifferent Constraint. In Georg Gottlob and Toby Walsh, editors, *Proceedings IJCAI 2003*, pages 245–250. Morgan Kaufmann, 2003. URL: `http://ijcai.org/Proceedings/03/Papers/036.pdf`.

28 Robert H. Storer, S. David Wu, and Renzo Vaccari. New Search Spaces for Sequencing Problems with Application to Job Shop Scheduling. *Management science*, 38(10):1495–1509, 1992.

29 Petr Vilím. $O(n \log n)$ Filtering Algorithms for Unary Resource Constraint. In *Proceedings of CPAIOR 2004*, pages 335–347, 2004.

30 Toby Walsh. Search in a Small World. In Thomas Dean, editor, *Proceedings of IJCAI 1999*, pages 1172–1177, 1999.

31 Young Su Yun. Genetic Algorithm with Fuzzy Logic Controller for Preemptive and Non-preemptive Job-shop Scheduling Problems. *Computers & Industrial Engineering*, 43(3):623–644, 2002.