International Journal of Production Research

ISSN: 0020-7543 (Print) 1366-588X (Online) Journal homepage: http://www.tandfonline.com/loi/tprs20

Parallel machine scheduling with tool loading: a constraint programming approach

Burak Gökgür, Brahim Hnich & Selin Özpeynirci

To cite this article: Burak Gökgür, Brahim Hnich & Selin Özpeynirci (2018): Parallel machine scheduling with tool loading: a constraint programming approach, International Journal of Production Research, DOI: 10.1080/00207543.2017.1421781

To link to this article: https://doi.org/10.1080/00207543.2017.1421781

Published online: 05 Jan 2018.

International Journal of Production Research, 2018
https://doi.org/10.1080/00207543.2017.1421781

Parallel machine scheduling with tool loading: a constraint programming approach

Burak Gökgür[a], Brahim Hnich[b] and Selin Özpeynirci[c*]

[a]*Graduate School of Business, Koç University, İstanbul, Turkey;* [b]*CES, ENIS, Sfax University & FSM, Monastir University, Monastir, Tunisia;* [c]*Department of Industrial Engineering, Izmir University of Economics, İzmir, Turkey*

(*Received 15 April 2016; accepted 11 December 2017*)

This paper presents constraint programming models that aim to solve scheduling and tool assignment problems in parallel machine environments. There are a number of jobs to be processed on parallel machines. Each job requires a set of tools, but limited number of tools are available in the system due to economic restrictions. The problem is to assign the jobs and the required tools to machines and to determine the schedule so that the makespan is minimised. Three constraint programming models are developed and compared with existing methods described in the literature.

Keywords: parallel machines; scheduling; constraint programming; makespan; tool loading

1. Introduction

Parallel machine scheduling has attracted attention from both academia and industry. In theory, it is a generalisation of single machine and a special case of flexible flow shop and the techniques developed for parallel machines can be used for multistage systems. In practice, resources in parallel are used in many real world environments (Pinedo 2002). Several problems may occur in manufacturing systems such as job assignment, scheduling, tool loading and tool switching. Tool loading is a complicating issue in scheduling problems since the number of tool copies are limited and may be smaller than the number of machines due to economic restrictions. Job scheduling and tool loading are important problems for industries in which there are large industrial machining centres and high technological manufacturing processes, such as CNC producers, and tooling and metal working industries (Shirazi and Frizelle 2001; Catanzaro, Gouveia, and Labbe 2015). For instance, as given in Crama et al. (1994) and Song and Hwang (2002), job and tool scheduling is an important problem for production systems that involves operations like forging, tooling, etc. Inefficient planning of job scheduling and tool loading may lead to under utilisation of capital intensive machines, and high level of machine idle time (Shirazi and Frizelle 2001). Therefore, efficient scheduling of jobs along with the tool loading problem enables a manufacturing system with parallel machines to increase its utilisation and decrease machines' idle times.

Crama (1997) defines combinatorial optimisation problems that may arise in automated manufacturing systems and provide mathematical models and solution approaches. Blazewicz and Finke (1994) provide a review of scheduling with resource management like tools, pallets, fixtures and automated guided vehicles. Edis, Oguz, and Ozkarahan (2013) present a review of studies on parallel machine scheduling with additional resources. There are a number of studies in the literature that combine job scheduling and resource assignment decisions. Roh and Kim (1997) discuss the tool loading and scheduling problems in flexible manufacturing systems. Specifically, they consider part and tool loadings, and part sequencing with the aim of minimising total tardiness, and propose heuristic approaches. Akturk and Ozkan (2001) develop a multistage algorithm to solve the integrated scheduling and tool management problems with the aim of minimising total production cost in a flexible manufacturing system. Ventura and Kim (2003) formulate the parallel machine scheduling problem as an integer programming model with the assumption that jobs may require additional resources. They find bounds using Lagrangean relaxation. Martin and White (2004) investigate a scheduling problem in which there are multiple candidates tools and machines to be selected to process a job, and they propose a new approach in the context of networks. Turkcan, Akturk, and Storer (2007) consider loading, scheduling and tool management problems in a setting where there are non-identical machines. With the objectives of minimising total manufacturing cost, and total weighted tardiness, they propose heuristic approaches. Chen and Chen (2008) consider a scheduling problem with flexible tooling resource constraints with the aim of coordinating part and machine scheduling. Recently, Zheng and Wang (2016) present a mixed integer programming model and propose a two-stage adaptive fruit fly optimisation approach for unrelated parallel machine scheduling problem with additional resource constraints. Fanjul-Peyro, Perea, and Ruiz (2017) propose two integer programming models and three

*Corresponding author. Email: selin.ozpeynirci@ieu.edu.tr

metaheuristic strategies for unrelated parallel machine scheduling problem in which the machines require a number of scarce resources while processing the jobs. Afzalirad and Rezaeian (2016) develop a new pure integer mathematical model with the objective of minimising makespan. They also propose two metaheuristic algorithms including genetic algorithm and artificial immune system. Ozpeynirci, Gokgur, and Hnich (2016) develop mathematical programming models for the loading and scheduling problem in parallel machine environments. Since the problem is NP-hard, as the problem size increases, the computation time increases rapidly and the optimal solution cannot be found even for moderate-sized instances. They also provide a heuristic solution based on tabu search approach and present the results. Ozpeynirci (2015) develops a time-indexed mathematical model for the same problem. Since the model is unable to solve even very small sized instances, they propose a heuristic algorithm based on the model. Monch et al. (2011) and Monch, Fowler, and Mason (2012) consider parallel machine scheduling problems with auxiliary resources in semiconductor manufacturing.

In this study, taking a constraint programming (CP) approach, we aim to develop more efficient solution approaches to the tool loading and job scheduling problem introduced in Ozpeynirci (2015). In the last few decades, efficient constraint-based scheduling algorithms have made constraint programming a method of choice for modelling and solving scheduling problems (e.g. Aggoun and Beldiceanu 1993; Caseau and Laburthe 1994; Baptiste and LePape 1995; Colombani 1996; Nuijten and Aarts 1996).

In recent years, constraint programming approaches have been developed for tool management and operation scheduling problems. Zeballos, Quiroga, and Henning (2010) propose a constraint programming approach considering a variety of constraints found in manufacturing environments such as machine eligibility and upper limits on costs. Zeballos (2010) include tool life, number of tools in the system and tool magazine capacities in their constraint programming model and present a search strategy that reduces the computation time. Edis and Ozkarahan (2011) study a parallel machine scheduling problem with machine eligibility restrictions, and they also develop a constraint programming model for the constrained problem. Novas and Henning (2014) propose a constraint programming model that takes into account machine loading, scheduling, parts routing, buffer scheduling, tool allocation and AGV scheduling.

First, we define the problem in the next section. In Sections 3 and 4, constraint programming models are defined and the experimental results are given, respectively. Finally, we conclude the paper in Section 5.

2. Problem definition

There are n jobs to be processed on m parallel machines. Each job must be assigned to exactly one machine and preemption is not allowed, i.e. once the processing of the job starts, it cannot stop until complete. Each machine is capable of processing each job; however, the processing times may vary. Let p_{ij} be the processing time of job i on machine j. In addition, job i requires a set of tools $l(i)$ to be processed. There are t tool types and r_k copies of tool k are available in the system. Each tool occupies one tool slot and the number of tools required by a job does not exceed the tool magazine capacity of a machine.

According to the notation used by Blazewicz et al. (2007), the problem under consideration can be denoted by $R_m|res\ tr_k1|C_{max}$ considering tools as resources. 'R_m' corresponds to m unrelated machines in parallel, while '$res\ tr_k1$' implies that there are t different resources; for each resource k a certain availability r_k exists and each job requires at most 1 unit of each resource.

Due to economic restrictions, the number of tool copies may be fewer than the number of machines. Therefore, if a tool required to process the next job is not loaded on the machine, it must either be taken from another machine or the tool crib. If all tool copies are in use by other machines, the process must be delayed until a copy becomes available. We assume that the time required for switching the tools is negligible. The assumptions about tool usage are given below:

- Tools are not shared between the machines, i.e. the loaded tools are not removed during the processing.
- Tools are moved between the machines whenever necessary, but tool switching times are negligible.
- Tool magazines of the machines are initially empty.
- Each tool occupies one tool slot.
- Tools never break down during processing.
- The number of tools required by an operation does not exceed the tool magazine capacity of the machines.

The problem is to schedule the jobs on parallel machines with their required tools so that the makespan is minimised.

3. Constraint programming models

In this section, we explore alternative CP formulations for our problem to the Mixed Integer Programming models. First, we give an overview of the constraint programming, and the scheduling abstractions and propagation algorithms in Sections 3.1 and 3.2, respectively. Then, we present our CP-based models in Sections 3.3–3.5.

3.1 *Constraint programming*

Combinatorial Optimisation problems are ubiquitous in industry (Tsang 1993). Examples are production planning subject to demand and resource availability so that profit is maximised, scheduling jobs on machines subject to precedence constraints so that the makespan is minimised, vehicle routing subject to initial and final location of the goods and the transportation vehicles so that delivery time and fuel expenses are minimized, etc. Such types of interesting problems arising from real life can be represented as *constraint satisfaction problems* (CSPs).

A CSP consists of a set of variables, each with a finite domain of values, and a set of constraints specifying allowed combinations of values for some variables (Tsang 1993). A *solution* to a CSP is an assignment of variables to values in their respective domains such that all of the constraints are satisfied.

Constraint propagation techniques are inference methods that help reducing the original CSP into another smaller one. The *size* of a CSP is the product of the domain sizes of all variables. To achieve this, each constraint in the original CSP is associated with a constraint propagator. The propagator removes *inconsistent* values from the domains of the variables in the scope of this constraint. Inconsistent values are values that do not appear in any solution to that particular constraint. Inconsistent values can be inferred by using a number of local consistency concepts. For example, a constraint c is *generalised arc consistent* (*GAC*), if, when a variable is assigned any of the values in its domain, there exist compatible values in the domains of all the other variables of c (Mohr and Masini 1988). A CSP is GAC if all constraints are GAC.

A solution to a CSP is an assignment of values to the variables satisfying the constraints. To find such solutions, constraint solvers often use tree search algorithms that construct partial assignments and enforce a local consistency, such as generalised arc consistency, to prune the search space. Enforcing a local consistency is traditionally called *constraint propagation*. One of the most commonly used local consistencies is generalised arc consistency. A constraint c is *generalised arc consistent* (GAC), if, when a variable in the scope of C is assigned any value in its domain, there exists an assignment to the other variables in C such that C is satisfied (Mohr and Masini 1988). This satisfying assignment is called *support* for the value. On binary constraints (those involving just two variables), generalised arc consistency is called arc consistency (AC). Generalised arc consistency is established on a constraint c by removing elements from the domains of variables in $vars(c)$ until the GAC property holds.

Another local consistency is *bound consistency* (*BC*), which requires the domains of the variables to be represented by an interval. For integer variables, the values can obey a natural total order, therefore the domain can be represented by an interval whose lower bound is the minimum value and the upper bound is the maximum value in the domain. Given a constraint C, a *bound support* on C is a tuple τ on $var(C)$ that assigns to each integer variable a value between its minimum and maximum. A value for an integer variable is *bound consistent with C* iff there exists a bound support assigning this value to this variable. A constraint C is *bound consistent* (*BC*) iff for each integer variable X_i, its minimum and maximum values belong to a bound support.

3.2 *Scheduling abstractions*

One of the most fundamental concepts for scheduling applications is the concept of an activity. An *activity* is an abstraction that can be thought of as an object containing three data items: a starting date; a duration; and an ending date, where the ending date is the starting date plus the duration. For instance, for the *activity A*, *start(A)* and *end(A)* represent the start time and the end time of *A*. The smallest values in the domains of *start(A)* and *end(A)* are called the earliest start time and the earliest end time of the *activity A* (EST_A and EET_A). Similarly, the greatest values in the domains of *start(A)* and *end(A)* are called the latest start time and the latest end time of the *activity A* (LST_A and LET_A). The duration of the activity is an additional variable, defined as the difference between the end time and the start time of the activity.

Another fundamental concept for scheduling applications is the type of resources. In scheduling problems, there are two types of resources: *unary* and *cumulative*. A *unary resource* is a disjunctive resource that models a set of non-interruptible activities which must not overlap in time. Once a unary resource starts processing an activity, it cannot stop or change the activity until the processing is finished. For example, if activities A and B require the same unary resource, they cannot intersect. In *cumulative resource*, a resource can be used in parallel, given that the resource capacity is not exceeded. For instance, the activities A and B use the same cumulative resource, provided that the amount of the resource assigned to these activities does not exceed the resource capacity.

There are many constraint propagation algorithms developed, according to type of resources. Le Pape, Baptiste, and Nuijten (2001) present an overview of constraint propagation techniques applied to different scheduling problems.

For the non-preemptive scheduling problem with cumulative resources, there are two constraint propagation techniques based on: timetable and disjunctive. Timetable propagation technique consists of maintaining bound consistency (Lhomme 1993) on the cumulative resource capacity formula, and using the maximal values of assigned capacity to compute the earliest and latest start and end times of activities (e.g. Le Pape 1994; Le Pape and Baptiste 1996, 1997). Disjunctive

Table 1. Parameter of the edge-finding example.

Activity	r_i	d_i	p_i	lst_i	eet_i
A	4	16	2	14	6
B	6	16	4	12	10
C	7	15	5	10	12

constraint propagation technique consists of maintaining bound consistency on the formula: $[end(A) \leq start(B)]$ or $[end(B) \leq start(A)]$.

There is another propagation technique applied to non-preemptive and preemptive scheduling problems with unary and cumulative resources, named as 'edge-finding' algorithm. For the non-preemptive scheduling problem with unary resources, the edge-finding algorithm consists of determining whether a given activity A must execute before (or after) a given set of activities. Two types of conclusions can be drawn: new ordering relations and new time bounds. Carlier and Pinson (1994) show that all the corresponding deductions can be done in $O(nlog(n))$, where n is the number of activities requiring the resource. For the non-preemptive scheduling problem with cumulative resources, the edge-finding algorithm consists of determining whether an activity A must start or end before (or after) a set of activities (adding edges in the graph representing the possible orderings of start and end times of activities). See (Nuijten 1994; Baptiste et al. 2006; Laborie et al. 2009), for a detailed discussion of the algorithm.

There are also other methods such as the 'Not-first' and 'Not-last', and energetic reasoning. 'Not-first' and 'Not-last' rules have been developed as a 'negative' counterpart to edge-finding rules. Energetic reasoning rules compute the minimal contribution of each activity to a given interval. For a detailed discussion of the algorithm, see (Le Pape, Baptiste, and Nuijten 2001).

Most CP languages provide such abstractions for representing activities and unary resources, but use different notations. For instance, IBM ILOG CP Optimizer 2.3 (IBM ILOG 2009) embed several constraint propagation algorithms and modelling constructs for scheduling applications. In this paper, however, rather than bind ourselves with the syntax of any specific CP language, we will keep the presentation at a high-level, and focus on the modelling decisions without becoming dependent on any particular CP language details and implementation. We will also only introduce the constraints needed for understanding our models and not cover all the scheduling constraints.

To define an activity variable, we propose the keyword `Activity`. For instance `Activity a` defines activity variable `a`. To refer to the starting time, ending time, and duration of activity `a`, we use `StartOf(a)`, `EndOf(a)`, and `DurationOf(a)`, respectively. If we wish to express that an activity `a` precedes another activity `b`, we use the keyword `Before` to state the constraint `a Before b`.

We will use the type keyword `UnaryResource` in order to declare a unary resource that models a set of non-interruptible activities which must not overlap in time. For instance, `UnaryResource M` defines a unary resource `M`. *Edge finding* algorithm is one of the most popular constraint propagation technique for disjunctive resources. Bartak (2003) presents an illustrative example of how the edge-finding mechanism is executed. In the example, there are three activities and the related parameters are given in Table 1. In the Table 1, r_i, lst_i, eet_i and d_i denote the release date, the latest start time, the earliest end time, and the deadline of activity A_i. A, B and C create three disjunctive constraints; 1) $((end(A) \leq start(B))||(end(B) \leq start(A)))$, 2) $((end(A) \leq start(C))||(end(C) \leq start(A)))$ and 3) $((end(B) \leq start(C))||(end(C) \leq start(B)))$. In the propagation of the first constraint, we cannot decide which activity is executed first, because $eet_A < lst_B$ and $eet_B < lst_A$. Similarly, in the propagation of the second constraint, since $eet_A < lst_C$ and $eet_C < lst_A$, the algorithm cannot determine the sequence of the activities A and C. The same results is obtained for the third constraint. As a result, the edge-finding algorithm deduce nothing for this specific example.

Some resources may have a specific capacity which can be shared by different activities. At any specific time, the total amount of the capacity used by all activities on such a resource does not exceed the resource capacity. These resources are *cumulative resources* that are defined using the keyword

```
DiscreteResource R(capacity(c))
```

where `R` is defined as a cumulative resource with discrete capacity `c`. (See Baptiste et al. 2006; Laborie et al. 2009, for a detailed discussion of propagation techniques for cumulative resources used in IBM ILOG CP Optimizer).

In many applications, an activity may require one of several *alternative* resources. That is, these resources are equivalent from the activity standpoint, but the activity must be assigned to exactly one of these alternative resources. As mentioned in Baptiste et al. (2006), *alternative* unary resource constraints are propagated as if the activity A_i were split into $|D_{altern(A_i)}|$

Objective function:
(1) Minimize $\texttt{max}\{\texttt{endOf}(J_1), \ldots, \texttt{endOf}(J_n)\}$
Decision variables:
(2) $\texttt{Activity}\ J_i, \quad \forall i \in 1, \ldots, n$
(3) $\texttt{UnaryResource}\ M_j, \quad \forall j \in 1, \ldots, m$
(4) $W_{ik} \in \{1, \ldots, r_k\}, \quad \forall i \in \{1, \ldots, n\}, \forall k \in \{1, \ldots, t\}$
Constraints:
(5) $\texttt{Alternative}(J_i, [M_1, \ldots, M_m], [p_{i1}, \ldots, p_{im}]), \forall i \in \{1, \ldots, n\}$
(6) $W_{ik} = W_{qk} \rightarrow (J_i\ \texttt{Before}\ J_q \vee J_q\ \texttt{Before}\ J_i), \forall \langle i, q, k \rangle \in S$

Figure 1. CP1: model based on modeling machines as unary resources.

fictive activities A_i^u where each activity A_i^u requires resource R_u. The alternative resource constraint maintains the following: $r_i = min\{r_i^u | u \in D_{altern(A_i)}\}$, $lst_i = max\{lst_i^u | u \in D_{altern(A_i)}\}$, $eet_i = min\{eet_i^u | u \in D_{altern(A_i)}\}$, $d_i = max\{d_i^u | u \in D_{altern(A_i)}\}$, $lb(duration(A_i)) = min\{lb(duration(A_i^u)) | u \in D_{altern(A_i)}\}$, and $ub(duration(A_i)) = max\{lb(duration(A_i^u)) | u \in D_{altern(A_i)}\}$ where r_i, lst_i, eet_i and d_i denote the release date, the latest start time, the earliest end time, and the deadline of activity A_i. Constraint propagation updates new bounds for alternative activities A_i^u on the alternative resource. When the bounds of an activity A_i^u turn out to be inconsistent, the resource is removed from the alternative resource set of activity.

To express that a list of resources `L` forms a pool of alternative resources for an activity `a`, it is possible to use the global `Alternative(a,L)` constraint. For instance, if an activity `a` requires one of the alternative unary resources `M1` or `M2`, one can impose the following constraint:

```
Alternative(a,[M1,M2]).
```

Furthermore, in some situations, the processing time of an activity depends on the assigned resource from the set of alternative ones. In such cases, we can use the `Alternative(a,L,P)` constraint with an extra optional list `P` to specify the processing time of activity `a` for each of the alternative resources in `L`. So, if `p1` is the processing time of activity `a` on resource `M1`, and `p2` is the processing time of activity `a` on resource `M2`, and `M1` and `M2` are alternative machines for activity `a`, we can state the following alternative constraint:

```
Alternative(a,[M1,M2],[p1,p2]).
```

If, instead, an activity requires a cumulative resource with a specific demand `d` from a set of alternative cumulative resources, we use the following version of the alternative constraint:

```
Alternative(a,d,[M1,M2]).
```

To check whether or not an activity `a` is assigned to a machine `M`, one can use the following constraint which is true when `a` is assigned to a machine `M`, false otherwise:

$$\texttt{hasActivity}(M, a)$$

The above abstractions have been implemented in various constraint solvers using different syntax and using various propagation algorithms. An activity i in constraint programming can be represented by three decision variables, for the start time s_i, for the duration d_i, and for the completion time e_i, with an extra constraint stating that $e_i = s_i + d_i$. However, in order to make constraint propagation tractable, an activity is simply represented as an interval $[est_i, lct_i - p_i]$ where est_i is the earliest possible starting time, lct_i is the latest possible completion time, and p_i is the processing time. The propagation algorithms in such a case contract this interval by tightening the bounds est_i and lct_i where holes in these intervals are ignored. Based on the interval representation for activities, several efficient propagation algorithms have been proposed to efficiently propagate the resource (i.e. disjunctive and cumulative constraints) as well as the other scheduling constraints (e.g. `Before`) by reasoning on intervals (Caseau and Laburthe 1994, 1995; Baptiste and LePape 1996; Martin and Shmoys 1996; Torres and Lopez 2000; Le Pape, Baptiste, and Nuijten 2001; Vilim 2004; Vilim, Bartak, and Cepek 2004; Mercier and Van Hentenryck 2008).

3.3 *Machines as unary resources*

In our problem, jobs can be represented as activities, and machines as unary resources. Each job can be assigned to any of the alternative machines. The CP model based on modelling jobs as activities and machines as unary resources is shown in

Objective function:
(1) Minimize $\texttt{max}\{\texttt{endOf}(J_1), \ldots, \texttt{endOf}(J_n)\}$
Decision variables:
(2) $\texttt{Activity}\ J_i, \quad \forall i \in 1, \ldots, n$
(3) $\texttt{UnaryResource}\ M_j, \quad \forall j \in 1, \ldots, m$
(4) $\texttt{UnaryResource}\ T_{kc}, \quad \forall k \in 1, \ldots, t, \forall c \in \{1, \ldots, r_k\}$
Constraints:
(5) $\texttt{Alternative}(J_i, [M_1, \ldots, M_m], [p_{i1}, \ldots, p_{im}]), \forall i \in \{1, \ldots, n\}$
(6) $\texttt{Alternative}(J_i, [T_{k1}, \ldots, T_{kr_k}]), \forall i \in \{1, \ldots, n\}, \forall k \in l(i)$

Figure 2. CP2: model based on machine and tool unary resources.

Figure 1. We refer to this model as CP1. In (2) we define an activity J_i for each job i. In (3) we define a unary resource M_j for each machine j. Constraints (5) impose an `Alternative` constraint for each job i specifying the set of alternative machines where the processing time is machine-dependent. The objective function (1) minimises the maximum ending time of all activities, i.e. minimising the makespan.

Indeed, using CP, we observe that we can compactly and elegantly represent the assignment of jobs to the parallel machines in a seamless way. What remains to be addressed, however, is the issue of tool conflicts. Recall that if two jobs use the same tool copy, then they must not overlap in time.

Let

$$S = \{\langle i, q, k\rangle | i \neq q, k \in l(i) \cap l(q), i \in \{1, \ldots, n\}, q \in \{1, \ldots, n\}\}$$

where $\langle i, q, k\rangle \in S$ iff job i and job q require the same tool k.

One way to model the tool conflict constraints is by introducing a decision variable as shown in (4) for each job i that requires a tool type k (W_{ik}) whose domain is the indices of the available copies of tool k. That is $W_{ik} = c$ iff job i uses copy c of tool k. Now, logical constraints (6) express that if job i and job q that require the same tool use the same tool copy ($W_{ik} = W_{qk}$), then either job i starts before job q or vice versa.

3.4 *Tools as unary resources*

The previous CP model uses logical constraints in order to enforce that two jobs requiring the same tool type and using the same copy must not overlap in time. These constraints are weak to propagate and may hinder constraint propagation. Another way to model this is by viewing each copy of every tool type as a unary resource itself. By doing so, each job that requires a particular tool type can be assigned to any of the alternative tool copy resources. In this way, we exploit again the full power of the global unary resource constraints to effectively and efficiently model the assignment of jobs to tool copies without violating the overlapping constraints. The CP2 model in Figure 2 no longer needs the W_{ik} variables which are replaced by unary resources, one for each copy of a given tool. Furthermore, instead of the logical constraints (6) of CP1, we rely on the unary resource global constraints as well as the alternative global constraint to express them more effectively and propagate them more efficiently.

3.5 *Breaking tool copy symmetry*

The copies of a particular tool are indistinguishable. However, in both CP models we distinguish between them, which introduces unnecessary symmetry into the models. The presence of symmetry in constraint models can cause a great deal of wasted search effort by exploring symmetrically equivalent regions of the search space. Therefore, by eliminating the symmetry we may significantly accelerate the search.

Cohen et al. (2006) define two types of symmetry in constraint programming. *Solution symmetry* is a permutation of variable-value pairs which preserves the set of solutions while *problem symmetry* is a permutation of variable-value pairs which preserves the set of constraints. Special cases of problem and/or solution symmetry are variable and value symmetry. A *variable symmetry* is a symmetry in which a set of variables can be freely permuted, whereas a *value symmetry* is a symmetry in which a set of values can be freely permuted. A further special case of variable symmetry is the so-called *index or matrix symmetry*, particularly where the variables of a matrix can be freely permuted row-wise and/or column-wise (Flener et al. 2002).

Several distinct methods have been reported for symmetry breaking in CP:

Reformulation: If we can reformulate the problem so that the model contains no symmetry, then there is clearly no need to break it. This is the ideal case whenever possible.

Static symmetry breaking : If we can add extra symmetry-breaking constraints to the model, which makes symmetric solutions infeasible, then we have broken the symmetry. A special case of this method is when we have index symmetry. For an $n \times m$ matrix with full row (or column) symmetry, all row (or column) symmetries can be broken by adding constraints to the model to lexicographically order rows (or columns) (Flener et al. 2002). Because of its power, low memory and runtime overhead and ease of use, the lexicographic ordering constraint `lex` is a popular way of eliminating row (or column) symmetry (Frisch et al. 2006).

Dynamic symmetry breaking: We have two dynamic methods for breaking symmetry during search. The first, known as *Symmetry Breaking During Search* (SBDS), was invented by Backofen and Will (1999) and developed by Gent and Smith (2000). In SBDS, constraints are added during search so that, after backtracking from a decision, future symmetrically equivalent decisions are disallowed. The second method known as *Symmetry Breaking by Dominance Detection* (SBDD) was independently invented by Brown, Finkelstein, and Purdom (1989), Fahle, Schamberger, and Sellmann (2001), Focacci and Milano (2001). The method was combined with GAP (Groups, Algorithms, Programming-a System for Computational Discrete Algebra (The GAP Group 2016)) to give GAP-SBDD (Gent et al. 2003). SBDD breaks all symmetries but does not add constraints before or during search. Instead it detects when the current search state is symmetrical to a previously-explored 'dominating' state, and backtracks before exploring that state.

In this paper, we choose to lexicographically order the columns using the global lex constraint because in (Flener et al. 2002) the authors show that decomposing the lex constraint into primitive constraints available in current finite-domain constraint toolkits either reduces the strength or increases the cost of constraint propagation. Indeed, the global lex constraint is one of the most effective and efficient ways of dealing with row/column symmetry, since it achieves the highest possible level of pruning in linear time, i.e. it removes all inconsistent values from the variable domains in linear time. We break all tool copy symmetry in two ways. In the first approach, we transform the tool copy symmetry into column symmetry through a reformulation, and then add statically symmetry-breaking constraints to the resulting models. The second approach reformulates the problem in such a way that the new model does not contain the tool copy symmetry.

We now present the details of the first approach. Consider the model CP1. For each tool k, the values of the decision variable W_{ik} constitutes the copy number of tool k used by job i. In any assignment, any permutation of these values of the jobs requiring the same tool will generate a symmetric, but essentially the same, solution.

The value symmetry in CP1 could be broken by transforming this symmetry into column symmetry in a matrix, and breaking all columns' symmetry using the lexicographic ordering constraints (Flener et al. 2002). To do this, for each tool k, we introduce a matrix of binary variables M_k indexed by the jobs requiring a copy of that tool and tool copy numbers. The meaning of $M_k[i, c] = 1$ is that job i uses tool copy c of tool k. This set of redundant decision variables need to be consistent with the W_{ik} variable assignment. Thus, we need to add the following channelling constraints:

$$W_{ik} = c \leftrightarrow M_k[i, c] = 1$$

Now, the columns of this matrix correspond to different tool copies. Hence, we can freely permute them without violating any of the problem constraints. To break column symmetry, we can lexicographically order these columns using the global `lex` constraint (Frisch et al. 2006).

Furthermore, we add the redundant sum constraint to strengthen constraint propagation that imposes that the sum of the elements in each row is exactly 1 (since each job requiring the tool must be assigned to one copy).

In CP2, we can perform exactly the same procedure. However, since rather than the W_{ik} variables, we have the unary resource T_{kc}, we impose a different set of channeling constraints:

$$\texttt{hasActivity}(T_{kc}, i) \leftrightarrow M_k[i, c] = 1$$

We have shown that models CP1 and CP2 can be extended to break all tool copies' symmetry by introducing the necessary matrix of decision variables for each tool, along with the proper channelling and global `lex` constraints. We refer to these extended models as CP1lex and CP2lex, respectively.

Our second approach to dealing with tool copy symmetry is through reformulation that eliminates the symmetry from the model. In fact, in CP2 we introduce a unary resource for each tool copy. This is the cause of the symmetry. So, if, for each tool we introduce instead a *cumulative resource* with a discrete capacity equal to the number of available copies, then

Objective function:
(1) Minimize $\texttt{max}\{\texttt{endOf}(J_1), \ldots, \texttt{endOf}(J_n)\}$
Decision variables:
(2) $\texttt{Activity}\ J_i, \quad \forall i \in 1, \ldots, n$
(3) $\texttt{UnaryResource}\ M_j, \quad \forall j \in 1, \ldots, m$
(4) $\texttt{DiscreteResource}\ T_k(capacity(r_k)), \quad \forall k \in 1, \ldots, t$
Constraints:
(5) $\texttt{Alternative}(J_i, [M_1, \ldots, M_m], [p_{i1}, \ldots, p_{im}]), \forall i \in \{1, \ldots, n\}$
(6) $\texttt{Alternative}(J_i, 1, T_k), \forall i \in \{1, \ldots, n\}, \forall k \in l(i)$

Figure 3. CP3: model based on tools as cumulative resources.

there is no symmetry at all. Figure 3 shows such a model which we refer to as CP3. In (4), we define a cumulative resource for each tool whose capacity is the number of available copies.

In (6), we enforce an `Alternative` constraint for each job requiring a tool copy by restricting the job to be assigned to the cumulative resource representing that tool in which the demand is set to 1.

In the proposed CP models, the main difference is that these models perform different levels of pruning during search depending on the different propagation algorithms associated with the different constraints and decision variables of each model. Thus, different search spaces are explored in different ways depending on each model yielding to differences in performance. What makes a model more effective than another is rather an art (Hnich 2003).The detailed version of all models (CP1, CP2 and CP3), coded in ILOG language, is presented in Appendix 1.

4. Experimental results

In order to measure the efficiency of above CP models, we conducted a series of experiments, in which the number of jobs was set to 8, 10, 15, 17, or 20. We set the number of machines to 2,3 or 4, and the number of tool types to 5 or 8. The number of tools in the set $l(i)$ was generated from a discrete uniform distribution between 2 and 5. The tools in set $l(i)$ were generated randomly. The value of r_k was randomly selected from the set {1,2}. Lastly, the p_{ij} values were generated from a discrete uniform distribution in the interval [25,150]. 10 problem instances were generated under each parameter setting. Results are provided for the combinations for which solutions could be found.

The models were solved by IBM OPL 12.5.1 using a Intel Core i5, 2.6 GHz computer with 8.00 GB RAM. We set an upper time limit of 1 h and we terminated the execution of the algorithm if the optimal solution could not be found in this limit. Also, the search type was set to default option (restart type) for each CP Model, which corresponds to depth-first search. After a specified number of failures was achieved, depth-first search was restarted.

4.1 *The effect of symmetry breaking*

We started by comparing the performances of CP1 and CP2 with and without symmetry breaking. Table 2 (and subsequent tables) give information for each class of instances (represented by $\langle n, m, t \rangle$ in which n, m, and t represent the number of jobs, machines, and tools, respectively). The information given consists of the number of optimal solutions (nbOptima) found by each model, the number of instances where a solution is found (not necessarily the optimal) in the time limit (nbSol) and the average solution time of the optima in seconds (time) and the objective function comparison of given models as a percentage deviation (% Imp. of *model name*). Also, information about the average number of branches and fails for all CP models is presented in Appendix 2.

Table 2 shows that CP1 can find the optimal solutions to the same problem instances as CP1lex. However, CP1lex can find solutions in some instances where CP1 cannot find any solution. Also, the solution quality of CP1lex is better than CP1. In fact, within the same time limits, CP1lex achieves better solutions, and according to Table 2, the average percentage improvement in the objective function is 15.95%. The experiments clearly show that symmetry breaking constraint enhances the search and allows CP1lex to solve more instances and find better solutions than CP1 within the same time limit.

Unlike CP1, as Table 3 shows, the effect of adding symmetry breaking constraints to CP2, rather than bring improvement in the objective function, instead only served to increase the computation overhead.

According to Table 3, CP2 can find the optimal solutions to the same instances as CP2lex. However, the solution time of CP2 is on average about 53% less than the CP2lex solution time.

It is quite interesting to observe that by declaring the tool copies as unary resources, the CP2 model becomes less sensitive to the tool copy symmetry and the effect of employing a better propagation algorithm to enforce the disjunctive constraints outweighs the effect of symmetry breaking constraints.

Table 2. Comparison of performances of CP1 and CP1 with symmetry breaking.

	CP1			CP1lex			
$\langle n, m, t\rangle$	nbOptima	nbSol	time	nbOptima	nbSol	time	% Imp. of CP1lex
⟨8, 2, 5⟩	0	10/10	–	0	10/10	–	0.00
⟨8, 2, 8⟩	3	10/10	734	3	10/10	734	0.00
⟨10, 2, 5⟩	0	10/10	–	0	10/10	–	0.00
⟨10, 2, 8⟩	0	10/10	–	0	10/10	–	0.00
⟨10, 3, 5⟩	0	8/10	–	0	10/10	–	0.00
⟨10, 3, 8⟩	0	9/10	–	0	10/10	–	0.00
⟨15, 2, 8⟩	0	10/10	–	0	10/10	–	4.99
⟨17, 2, 5⟩	0	10/10	–	0	10/10	–	11.52
⟨17, 2, 8⟩	0	10/10	–	0	10/10	–	5.84
⟨17, 3, 5⟩	0	6/10	–	0	10/10	–	14.69
⟨17, 3, 8⟩	0	8/10	–	0	10/10	–	17.01
⟨17, 4, 5⟩	0	8/10	–	0	10/10	–	29.53
⟨17, 4, 8⟩	0	1/10	–	0	10/10	–	18.93
⟨20, 2, 5⟩	0	10/10	–	0	10/10	–	15.58
⟨20, 2, 8⟩	0	10/10	–	0	10/10	–	14.60
⟨20, 3, 5⟩	0	5/10	–	0	10/10	–	22.43
⟨20, 3, 8⟩	0	3/10	–	0	10/10	–	20.36

Table 3. Comparison of performances of CP2 and CP2 with symmetry breaking.

	CP2			CP2lex			
$\langle n, m, t\rangle$	nbOptima	nbSol	time	nbOptima	nbSol	time	% Imp. of CP2lex
⟨8, 2, 5⟩	6	10/10	2.90	6	10/10	3.76	0.00
⟨8, 2, 8⟩	4	10/10	3.82	4	10/10	4.47	0.00
⟨10, 2, 5⟩	9	10/10	3.00	9	10/10	4.30	0.00
⟨10, 2, 8⟩	4	10/10	3.37	4	10/10	4.14	0.00
⟨10, 3, 5⟩	8	10/10	2.50	8	10/10	358	0.00
⟨10, 3, 8⟩	6	10/10	2.88	6	10/10	5.10	0.00
⟨15, 2, 8⟩	3	10/10	2.54	3	10/10	11.22	0.00
⟨17, 2, 5⟩	6	10/10	0.93	6	10/10	4.66	0.00
⟨17, 2, 8⟩	3	10/10	5.27	3	10/10	6.21	0.00
⟨17, 3, 5⟩	6	10/10	3.35	6	10/10	3.93	0.00
⟨17, 3, 8⟩	5	10/10	4.71	5	10/10	8.00	0.00
⟨17, 4, 5⟩	7	10/10	3.32	7	10/10	4.21	0.00
⟨17, 4, 8⟩	7	10/10	4.07	7	10/10	7.06	0.00
⟨20, 2, 5⟩	2	10/10	3.26	2	10/10	4.61	0.00
⟨20, 2, 8⟩	2	10/10	11.39	2	10/10	48.51	0.00
⟨20, 3, 5⟩	5	10/10	5.03	5	10/10	3.90	0.00
⟨20, 3, 8⟩	5	10/10	1.58	5	10/10	5.87	0.00

Thus, from these experiments, we can conclude that CP1lex is superior to CP1 in terms of the number of instances where a solution is found, and the solution quality, whereas CP2 is slightly better than CP2lex in terms of solution time.

4.2 *Comparing CP models*

The previous set of experiments show that CP1lex and CP2 are better than CP1 and CP2lex, respectively. Now, we compare the performance of CP1lex, CP2 and CP3.

Table 4 compares CP2 and CP1lex. According to Table 4, CP2 can clearly find more optimal solutions than CP1lex. Furthermore, the average solution time of CP2 is prominently less than CP1lex. In fact, the improvement in terms of running

Table 4. Comparison of performances of CP2 and CP1lex.

	P2			CP1lex			
$\langle n, m, t \rangle$	nbOptima	nbSol	time	nbOptima	nbSol	time	% Imp. of CP2
⟨8, 2, 5⟩	6	10/10	2.90	0	10/10	–	0.00
⟨8, 2, 8⟩	4	10/10	3.82	3	10/10	734	0.00
⟨10, 2, 5⟩	9	10/10	3.00	0	10/10	–	0.00
⟨10, 2, 8⟩	4	10/10	3.37	0	10/10	–	0.00
⟨10, 3, 5⟩	8	10/10	2.50	0	10/10	–	0.00
⟨10, 3, 8⟩	6	10/10	2.88	0	10/10	–	0.00
⟨15, 2, 8⟩	3	10/10	2.54	0	10/10	–	0.00
⟨17, 2, 5⟩	6	10/10	0.93	0	10/10	–	0.00
⟨17, 2, 8⟩	3	10/10	5.27	0	10/10	–	0.00
⟨17, 3, 5⟩	6	10/10	3.35	0	10/10	–	0.00
⟨17, 3, 8⟩	5	10/10	4.71	0	10/10	–	0.00
⟨17, 4, 5⟩	7	10/10	3.32	0	10/10	–	0.00
⟨17, 4, 8⟩	7	10/10	4.07	0	10/10	–	1.58
⟨20, 2, 5⟩	2	10/10	3.26	0	10/10	–	0.00
⟨20, 2, 8⟩	2	10/10	11.39	0	10/10	–	0.00
⟨20, 3, 5⟩	5	10/10	5.03	0	10/10	–	0.25
⟨20, 3, 8⟩	5	10/10	1.58	0	10/10	–	0.27

Table 5. Comparison of performances of CP2 and CP3.

	CP2			CP3			
$\langle n, m, t \rangle$	nbOptima	nbSol	time	nbOptima	nbSol	time	% Imp. of CP2
⟨8, 2, 5⟩	6	10/10	2.90	6	10/10	2.57	0.00
⟨8, 2, 8⟩	4	10/10	3.82	4	10/10	2.50	0.00
⟨10, 2, 5⟩	9	10/10	3.00	9	10/10	3.27	0.00
⟨10, 2, 8⟩	4	10/10	3.37	4	10/10	2.80	0.00
⟨10, 3, 5⟩	8	10/10	2.50	8	10/10	2.87	0.00
⟨10, 3, 8⟩	6	10/10	2.88	6	10/10	2.60	0.00
⟨15, 2, 8⟩	3	10/10	2.54	3	10/10	3.65	0.00
⟨17, 2, 5⟩	6	10/10	0.93	6	10/10	20.9	0.00
⟨17, 2, 8⟩	3	10/10	5.27	3	10/10	5.99	0.00
⟨17, 3, 5⟩	6	10/10	3.35	6	10/10	6.19	0.00
⟨17, 3, 8⟩	5	10/10	4.71	5	10/10	6.31	0.00
⟨17, 4, 5⟩	7	10/10	3.32	7	10/10	5.80	0.00
⟨17, 4, 8⟩	7	10/10	4.07	7	10/10	6.53	0.00
⟨20, 2, 5⟩	2	10/10	3.26	2	10/10	3.09	0.00
⟨20, 2, 8⟩	2	10/10	11.39	2	10/10	48.5	0.00
⟨20, 3, 5⟩	5	10/10	5.03	5	10/10	5.31	0.00
⟨20, 3, 8⟩	5	10/10	1.58	5	10/10	6.38	0.00

times reaches up to 3 orders of magnitude. Moreover, objective function values obtained by CP2 are on average 0.12% less than that of CP1lex.

A possible explanation for the prevalence of CP2 over CP1lex is that by modelling each tool copy as a unary resource, the effect of the global disjunctive constraints outweighs the effect of symmetry-breaking constraints of CP1lex, which suffers from the logical constraints that cause weak propagation.

The final comparison between CP model is among CP2 and CP3 shown in Table 5.

According to Table 5, CP2 finds the same optimal solutions as CP3 except for the instance in the set(⟨20, 3, 8⟩). However, the solution time of CP2 is on average 53% less than CP3.

We again observe the prevalence of CP2 even though, in CP3, we do not have tool copy symmetry, and we use a cumulative constraint which replaces a number of disjunctive constraints in CP2. Indeed, it is NP-Complete to decide

Table 6. Comparison of performances of CP2 and MIP-Sym.

	CP2			MIP-Sym			
$\langle n, m, t \rangle$	nbOptima	nbSol	time	nbOptima	nbSol	time	% Imp. of CP2
⟨8, 2, 5⟩	10	10/10	1249.13	10	10/10	141.95	0.00
⟨8, 2, 8⟩	10	10/10	1427.76	10	10/10	14.63	0.00
⟨10, 2, 5⟩	10	10/10	679.16	10	10/10	911.70	0.00
⟨10, 2, 8⟩	10	10/10	3220.68	10	10/10	183.19	0.00
⟨10, 3, 5⟩	10	10/10	1344.52	10	10/10	689.87	0.00
⟨10, 3, 8⟩	10	10/10	1307.72	10	10/10	120.35	0.00
⟨15, 2, 8⟩	10	10/10	4359.39	5	10/10	5244.00	0.00
⟨17, 2, 5⟩	10	10/10	2744.73	0	10/10	–	0.00
⟨17, 2, 8⟩	10	10/10	4018.09	0	10/10	–	0.00
⟨17, 3, 5⟩	10	10/10	2331.23	0	10/10	–	0.00
⟨17, 3, 8⟩	10	10/10	2917.55	0	10/10	–	0.02
⟨17, 4, 5⟩	10	10/10	1854.49	0	10/10	–	0.22
⟨17, 4, 8⟩	10	10/10	1795.57	0	10/10	–	0.05
⟨20, 2, 5⟩	10	10/10	9940.14	0	10/10	–	0.25
⟨20, 2, 8⟩	10	10/10	4762.52	0	10/10	–	0.03
⟨20, 3, 5⟩	10	10/10	3229.14	0	10/10	–	1.55
⟨20, 3, 8⟩	10	10/10	2828.47	0	10/10	–	2.02

Table 7. Comparison of performances of CP2 and Tabu Search.

	CP2			Tabu Search			
$\langle n, m, t \rangle$	nbOptima	nbSol	time	nbOptima	nbSol	time	% Imp. of CP2
⟨8, 2, 5⟩	6	10/10	2.90	5	10/10	107.06	0.30
⟨8, 2, 8⟩	4	10/10	3.82	2	10/10	112.51	2.84
⟨10, 2, 5⟩	9	10/10	3.00	9	10/10	208.91	0.00
⟨10, 2, 8⟩	4	10/10	3.37	1	10/10	191.47	3.22
⟨10, 3, 5⟩	8	10/10	2.50	8	10/10	–	0.00
⟨10, 3, 8⟩	6	10/10	2.88	4	10/10	229.99	2.08
⟨15, 2, 8⟩	3	10/10	2.54	0	10/10	–	5.29
⟨17, 2, 5⟩	6	10/10	0.93	5	10/10	–	0.18
⟨17, 2, 8⟩	3	10/10	5.27	0	10/10	–	6.47
⟨17, 3, 5⟩	6	10/10	3.35	6	10/10	–	0.04
⟨17, 3, 8⟩	5	10/10	4.71	2	10/10	–	4.57
⟨17, 4, 5⟩	7	10/10	3.32	4	10/10	–	5.10
⟨17, 4, 8⟩	7	10/10	4.07	1	10/10	–	12.03
⟨20, 2, 5⟩	2	10/10	3.26	2	10/10	–	0.10
⟨20, 2, 8⟩	2	10/10	11.39	0	10/10	–	5.31
⟨20, 3, 5⟩	5	10/10	5.03	4	10/10	–	0.58
⟨20, 3, 8⟩	5	10/10	1.58	0	10/10	–	4.29

whether the cumulative constraint is satisfiable. A possible explanation is therefore that the implementation of the cumulative constraints of CP3 does not achieve significantly more pruning than the decomposition into disjunctive constraints in CP2.

Tables B1–B3 in Appendix 2 provide the number of branches and fails for all CP models, which also show the superiority of CP2. As a summary, CP2 seems to be the best because, of all CP models, it is the most evenly balanced.

4.3 *Comparing best CP model to other approaches*

In Table 6, we compare CP2 and the mathematical model with symmetry-breaking (MIP-Sym) developed in Ozpeynirci, Gokgur, and Hnich (2016). For this comparison, we set the upper time limit of 5 h. This is because MIP-Sym cannot find any solution to any instance in which the number of jobs is larger than 15. We compare the CP incumbent with the MIP-Sym lower bound solution, in the case that the optimal solution cannot be found by MIP-Sym.

Table 6 indicates that MIP-Sym performs better than CP2 in terms of execution time for instances in which the number of jobs is smaller or equal to 10. However, the results show that CP2 outperforms MIP-Sym in terms of number of optimal solutions found, as well as in terms of running times for larger instances. Specifically, CP2 performs better than MIP-Sym in terms of solution quality and execution time for instances in which the number of jobs is larger than 10. CP2 outperforms MIP-Sym in terms of number of optimal solutions found as well as in terms of running times for larger instances. MIP-Sym fails to find any solution to any instance in which the number of jobs is larger than 15. Within sets for which MIP-Sym finds a solution, CP2 finds better solutions to 0.09% of the instances, and average improvement in the objective function is 0.24% for instances in which CP2 finds better solutions than MIP-Sym. Also, it is worth noting that MIP-Sym cannot guarantee the optimality of solutions in some instances; however it finds the same solution as CP2, proven to be optimal by this model. For this reason, in sets $\langle 17, 2, 5\rangle$, $\langle 17, 2, 8\rangle$ and $\langle 17, 3, 5\rangle$, we report the percentage improvement as 0%, although the number of optimal solutions (nbOptima) for the MIP-Sym is zero.

Our final set of experiments compares CP2 against the Tabu search algorithm developed in Ozpeynirci, Gokgur, and Hnich (2016). We use results of CP2 achieved within 1 h in the comparison. The results of the comparison are shown in Table 7.

Based on the results of Table 7, CP2 again outperforms the Tabu search algorithm. Although Tabu search finds solutions to all instances, CP2 finds more optimal solutions. Furthermore, CP2 is significantly faster than Tabu search. The deviation of Tabu from the best obtained solution is on average 3.08%. Thus, CP2 finds better quality solutions in less time than Tabu search.

As a result of the experimental studies and comparisons reported above, CP2 outperforms mathematical programming, Tabu search and other CP approaches.

5. Conclusion

Ozpeynirci, Gokgur, and Hnich (2016) develop mathematical programming models and a Tabu search algorithm for the loading and scheduling problem in parallel machine environments. In this paper, we make a number of contributions that improve the computational efficiency of the approaches in Ozpeynirci, Gokgur, and Hnich (2016). By taking a constraint programming approach, we propose a number of alternative models that make use of powerful global constraints dedicated to modelling and solving scheduling problems. By effectively capturing the problem constraints, these global constraints produce more elegant, compact, and easy to read models. Furthermore, these global constraints enhance constraint propagation resulting in more efficient models than the mathematical programming or Tabu search counterpart. Finally, we present two ways of dealing with the tool copy symmetry, and analyse their respective effects. This study yields the following practical (managerial) insights:

- We propose an easy-to-read and worth-to-implement constraint programming formulation that allows a decision maker to solve the planning and scheduling problems of a manufacturing system efficiently.
- The proposed constraint programming approach obtains promising (optimal or nearly optimal) solutions in a shorter time which is reasonable for daily operations. It enables the decision-maker to respond easily to the changes in a manufacturing system.
- We show that the proposed constraint programming approach could obtain promising solutions, while traditional approaches, such as mixed-integer programming, could not in a manufacturing system where the number of jobs and/or machines is relatively high. Therefore, a decision-maker could use the proposed solution approach to generate efficient schedule plans of jobs and tools.
- In a manufacturing process, the efficient use of proposed constraint programming approach could ensure that the machines' idle time would be low and the productivity level would increase. Because, the proposed solution approach obtains optimal (or nearly optimal) scheduling plan of jobs and tools almost for all problem instances. It implies that a decision maker could complete the jobs in a shorter time and receive more jobs resulting from the efficient use of resources.

In the future, we aim to try to extend our method with more specialised labelling strategy as well considering adding more side constraints, such as tool and/or machine failures, to the problem. Also, other optimal solution techniques, such as branch and bound algorithm can be used to find efficient solutions to the problem under consideration. Another interesting future research direction would be to develop efficient algorithms for scheduling problem considering tool switching times.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This work is supported by the Scientific and Technological Research Council of Turkey (TUBITAK) [project number 110M492].

References

Afzalirad, M., and J. Rezaeian. 2016. "Resource-constrained Unrelated Parallel Machine Scheduling Problem with Sequence Dependent Setup Times, Precedence Constraints and Machine Eligibility Restrictions." *Computers & Industrial Engineering* 98: 40–52.

Aggoun, A., and N. Beldiceanu. 1993. "Extending CHIP in Order to Solve Complex Scheduling and Placement Problems." *Mathematical and Computer Modelling* 17 (7): 57–73.

Akturk, M., and S. Ozkan. 2001. "Integrated Scheduling and Tool Management in Flexible Manufacturing Systems." *International Journal of Production Research* 39 (12): 2697–2722.

Backofen, R., and S. Will. 1999. "Excluding Symmetries in Constraint-based Search." In *Principles and Practice of Constraint Programming-CP'99*. Vol. 1713 of *Lecture Notes in Computer Science*, edited by J. Jaffar, 73–87. Berlin.

Baptiste, P., and C. LePape. 1995. "A Theoretical and Experimental Comparison of Constraint Propagation Techniques for Disjunctive Scheduling." In *IJCAI-95 – Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, Vols 1 and 2, International Joint Conference on Artificial Intelligence*, Montreal, Canada, edited by C. S. Mellish, 600–606.

Baptiste, P., and C. LePape. 1996. "Edge-finding Constraint Propagation Algorithms for Disjunctive and Cumulative Scheduling." In *Scheduling, Proceedings 15th Workshop of the U.K. Planning Special Interest Group*, Liverpool.

Baptiste, P., P. Laborie, C. Le Pape, and W. Nuijten. 2006. "Constraint-based Scheduling and Planning." In *Handbook of Constraint Programming*, edited by F. Rossi, P. van Beek, and T. Walsh, 761–799. New York: Elsevier Science Inc.

Bartak, R. 2003. "Constraint-based Scheduling: An Introduction for Newcomers." In *Intelligent Manufacturing Systems 2003, IFAC Workshop Series*, edited by L. Monostori, B. Kadar and G. Morel, 69–74. Budapest: Elsevier.

Blazewicz, J., K. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. 2007. *Handbook on Scheduling: Models and Methods for Advanced Planning (International Handbooks on Information Systems)*. New Jersey: Springer-Verlag New York Inc.

Blazewicz, J., and G. Finke. 1994. "Scheduling with Resource-management in Manufacturing Systems." *European Journal of Operational Research* 76 (1): 1–14.

Brown, C., L. Finkelstein, and P. Purdom. 1989. "Backtrack Searching in the Presence of Symmetry." *Lecture Notes in Computer Science* 357: 99–110.

Carlier, J., and E. Pinson. 1994. "Adjustment of Heads and Tails for the Job-shop Problem." *European Journal of Operational Research* 78 (2): 146–161.

Caseau, Y., and F. Laburthe. 1994. "Improved CLP Scheduling with Task Intervals." In *Logic Programming: Proceedings of the Eleventh International Conference on Logic Programming, MIT Press Series in Logic Programming*, Santa Margherita, Italy, edited by P. VanHentenryck, 369–383.

Caseau, Y., and F. Laburthe. 1995. *Disjunctive Scheduling with Task Intervals*. Technical report, LIENS Technical Report, Ecole Normale Suprieure Paris.

Catanzaro, D., L. Gouveia, and M. Labbe. 2015. "Improved Integer Linear Programming Formulations for the Job Sequencing and Tool Switching Problem." *European Journal of Operational Research* 244 (3): 766–777.

Chen, J., and F. F. Chen. 2008. "Adaptive Scheduling and Tool Flow Control in Flexible Job Shops." *International Journal of Production Research* 46 (15): 4035–4059.

Cohen, D., P. Jeavons, C. Jefferson, K. E. Petrie, and B. M. Smith. 2006. "Symmetry Definitions for Constraint Satisfaction Problems." *Constraints* 11 (2–3): 115–137.

Colombani, Y. 1996. "Constraint Programming: An Efficient and Practical Approach to Solving the Job-shop Problem." In *Principles and Practice of Constraint Programming – CP96: Second International Conference, CP96 Cambridge, Proceedings*. Berlin: Springer.

Crama, Y. 1997. "Combinatorial Optimization Models for Production Scheduling in Automated Manufacturing Systems." *European Journal of Operational Research* 99 (1): 136–153.

Crama, Y., A. W. J. Kolen, A. G. Oerlemans, and F. C. R. Spieksma. 1994. "Minimizing the Number of Tool Switches on a Flexible Machine." *International Journal of Flexible Manufacturing Systems* 6 (1): 33–54.

Edis, E. B., C. Oguz, and I. Ozkarahan. 2013. "Parallel Machine Scheduling with Additional Resources: Notation, Classification, Models and Solution Methods." *European Journal of Operational Research* 230 (3): 449–463.

Edis, E. B., and I. Ozkarahan. 2011. "A Combined Integer/Constraint Programming Approach to a Resource-constrained Parallel Machine Scheduling Problem with Machine Eligibility Restrictions." *Engineering Optimization* 43 (2): 135–157.

Fahle, T., S. Schamberger, and M. Sellmann. 2001. "Symmetry Breaking." In *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming, CP '01*, 93–107. Paphos: Springer-Verlag.

Fanjul-Peyro, L., F. Perea, and R. Ruiz. 2017. "Models and Matheuristics for the Unrelated Parallel Machine Scheduling Problem with Additional Resources." *European Journal of Operational Research* 260 (2): 482–493.

Flener, P., A. M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. 2002. "Breaking Row and Column Symmetries in Matrix Models." In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming, CP '02*, 462–476. Ithaca, NY: Springer-Verlag.

Focacci, F., and M. Milano. 2001. *Global Cut Framework for Removing Symmetries*, 77–92. Berlin: Springer.

Frisch, A. M., B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh. 2006. "Propagation Algorithms for Lexicographic Ordering Constraints." *Artificial Intelligence* 170 (10): 803–834.

Gent, I., W. Harvey, T. Kelsey, and S. Linton. 2003. "Generic SBDD Using Computational Group Theory." In *Principles and Practice of Constraint Programming – CP 2003, Proceedings*, Kinsale, Ireland. Vol. 2833 of *Lecture Notes in Computer Science*, edited by F. Rossi, 333–347.

Gent, I., and B. Smith. 2000. "Symmetry Breaking in Constraint Programming." In *ECAI 2000: 14th European Conference on Artificial Intelligence, Proceedings*, Berlin, Germany. Vol. 54 of *Frontiers in Artificial Intelligence and Applications*, edited by W. Horn, 599–603.

Hnich, B. 2003. "Function Variables for Constraint Programming." PhD thesis, Uppsala University, Uppsala, Sweden.

IBM ILOG. 2009. *IBM ILOG OPL V6.3 Language User's Manual*.

Laborie, P., J. Rogerie, P. Shaw, and P. Vilím. 2009. "Reasoning with Conditional Time-intervals. Part II: An Algebraical Model for Resources." In *Proceedings of the Twenty-Second International Florida Artificial Intelligence Research Society Conference*, Florida.

Le Pape, C. 1994. "Implementation of Resource Constraints in ILOG SCHEDULE: A Library for the Development of Constraint-based Scheduling Systems." *Intelligent Systems Engineering* 3 (2): 55–66.

Le Pape, C., and Ph. Baptiste. 1996. "Constraint Propagation Techniques for Disjunctive Scheduling: The Preemptive Case." In *12th European Conference on Artificial Intelligence, Budapest, Hungary, August 11-16, 1996, Proceedings*, 619–623.

Le Pape, C., and Ph. Baptiste. 1997. "A Constraint Programming Library for Preemptive and Non-Preemptive Scheduling." In *Proceedings of 3rd International Conference and Exhibition on the Practical Application of Constraint Technology, The Practical Application Company*, London, 237–256.

Le Pape, C., P. Baptiste, and W. Nuijten. 2001. *Constraint-based Scheduling: Applying Constraint Programming to Scheduling Problems*. Hingham, MA: Kluwer Academic.

Lhomme, O. 1993. "Consistency Techniques for Numeric CSPS." In *IJCAI-93, Vols 1 and 2: Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Chambery, edited by R. Bajcsy, 232–238.

Martin, P., and D. B. Shmoys. 1996. *A New Approach to Computing Optimal Schedules for the Job-shop Scheduling Problem*, 389–403. Berlin: Springer.

Martin, R., and C. White. 2004. "Finite Capacity Scheduling with Multiple Tools and Machines." *International Journal of Production Research* 42 (8): 1557–1597.

Mercier, L., and P. Van Hentenryck. 2008. "Edge Finding for Cumulative Scheduling." *INFORMS Journal on Computing* 20 (1): 143–153.

Mohr, R., and G. Masini. 1988. "Good Old Discrete Relaxation." In *Proceedings of the 8th European Conference on Artificial Intelligence, ECAI'88*, 651–656. London/Boston: Pitman.

Monch, L., J. W. Fowler, S. Dauzere-Peres, S. J. Mason, and O. Rose. 2011. "A Survey of Problems, Solution Techniques, and Future Challenges in Scheduling Semiconductor Manufacturing Operations." *Journal of Scheduling* 14 (6): 583–599.

Monch, L., J. W. Fowler, and S. J. Mason. 2012. *Production Planning and Control for Semiconductor Wafer Fabrication Facilities: Modeling, Analysis, and Systems. Operations Research/Computer Science Interfaces Series*. Dordrecht: Springer.

Novas, J. M., and G. P. Henning. 2014. "Integrated Scheduling of Resource-constrained Flexible Manufacturing Systems Using Constraint Programming." *Expert Systems with Applications* 41 (5): 2286–2299.

Nuijten, W. 1994. "Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach." PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands.

Nuijten, W., and E. Aarts. 1996. "A Computational Study of Constraint Satisfaction for Multiple Capacitated Job Shop Scheduling." *European Journal of Operational Research* 90 (2): 269–284.

Ozpeynirci, S. 2015. "A Heuristic Approach Based on Time-indexed Modelling for Scheduling and Tool Loading in Flexible Manufacturing Systems." *International Journal of Advanced Manufacturing Technology* 77 (5–8): 1269–1274.

Ozpeynirci, S., B. Gokgur, and B. Hnich. 2016. "Parallel Machine Scheduling with Tool Loading." *Applied Mathematical Modelling* 40 (9–10): 5660–5671.

Pinedo, M. 2002. *Scheduling Theory, Algorithms and Systems*. New York: Prentice Hall.

Roh, H., and Y. Kim. 1997. "Due-date Based Loading and Scheduling Methods for a Flexible Manufacturing System with an Automatic Tool Transporter." *International Journal of Production Research* 35 (11): 2989–3003.

Shirazi, R., and G. Frizelle. 2001. "Minimizing the Number of Tool Switches on a Flexible Machine: An Empirical Study." *International Journal of Production Research* 39 (15): 3547–3560.

Song, C., and H. Hwang. 2002. "Optimal Tooling Policy for a Tool Switching Problem of a Flexible Machine with Automatic Tool Transporter." *International Journal of Production Research* 40 (4): 873–883.

The GAP Group. 2016. *GAP – Groups, Algorithms, Programming, Version 4.8.3*. http://www.gap-system.org/.

Torres, P., and P. Lopez. 2000. "On Not-first/Not-last Conditions in Disjunctive Scheduling." *European Journal of Operational Research* 127 (2): 332–343.

Tsang, E. 1993. *Foundations of Constraint Satisfaction*. Computation in Cognitive Science. London and San Diego: Academic Press.

Turkcan, A., M. S. Akturk, and R. H. Storer. 2007. "Due Date and Cost-based FMS Loading, Scheduling and Tool Management." *International Journal of Production Research* 45 (5): 1183–1213.

Ventura, J., and D. Kim. 2003. "Parallel Machine Scheduling with Earliness-tardiness Penalties and Additional Resource Constraints." *Computers & Operations Research* 30 (13): 1945–1958.

Vilim, P. 2004. "O (n log n) Filtering Algorithms for Unary Resource Constraint." In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Vol. 3011 of *Lecture Notes in Computer Science*, Berlin, edited by J. C. Regin and M. Rueher, 335–347.

Vilim, P., R. Bartak, and O. Cepek. 2004. "Unary Resource Constraint with Optional Activities." In *Principles and Practice of Constraint Programming – CP 2004, Proceedings*. Vol. 3258 of *Lecture Notes in Computer Science*, Berlin, edited by M. Wallace, 62–76.

Zeballos, L. J. 2010. "A Constraint Programming Approach to Tool Allocation and Production Scheduling in Flexible Manufacturing Systems." *Robotics and Computer-Integrated Manufacturing* 26 (6, SI): 725–743.

Zeballos, L. J., O. D. Quiroga, and G. R. Henning. 2010. "A Constraint Programming Model for the Scheduling of Flexible Manufacturing Systems with Machine and Tool Limitations." *Engineering Applications of Artificial Intelligence* 23 (2): 229–248.

Zheng, X., and L. Wang. 2016. "A Two-stage Adaptive Fruit Fly Optimization Algorithm for Unrelated Parallel Machine Scheduling Problem with Additional Resource Constraints." *Expert Systems with Applications* 65: 28–39.

Appendix 1

Objective function:
(1) minimize max(i in job) endOf(JobMachine[i])
Decision variables:
(2) interval JobMachine
(3) sequence machines
(4) W[i in job][k in tool]
Constraints:
(5) $\forall i \in \{1,\ldots,n\}$ alternative(JobMachine[i],all(w in ⟨i,j⟩: w.i==i) optionalJobMachine[w])
(6) $\forall \langle i,q,k\rangle \in S$ (W[i][k]==W[q][k]) =>
((startOf(JobMachine[i]) >= endOf(JobMachine[q]))
||
(startOf(JobMachine[q]) >= endOf(JobMachine[i])))
(7) $\forall i,q \in \{1,\ldots,n\}$
((startOf(JobMachine[i]) >= endOf(JobMachine[q]))
||
(startOf(JobMachine[q]) >= endOf(JobMachine[i])));

Figure A1. CP1: model based on modeling machines as unary resources.

Objective function:
(1) minimize max(i in job) endOf(JobMachine[i])
Decision variables:
(2) interval JobMachine
(3) sequence machines
(4) sequence tools
Constraints:
(5) $\forall i \in \{1,\ldots,n\}$ alternative(JobMachine[i],all(w in ⟨i,j⟩: w.i==i) optionalJobMachine[w])
(6) $\forall i \in \{1,\ldots,n\}\ \forall k \in l(i)$
alternative(JobMachine[i],all(r in ⟨i,k⟩: r.i==i && r.k == k) optionalJobToolCopy[r])
(7) $\forall j \in \{1,\ldots,m\}$ noOverlap(machines[j]);
(8) $\forall k,c \in \{1,\ldots,t\},\{1,\ldots,r_k\}$ noOverlap(tools[k,c]);

Figure A2. CP2: model based on machine and tool unary resources.

Objective function:
(1) minimize max(i in job) endOf(JobMachine[i])
Decision variables:
(2) interval JobMachine
(3) sequence machines
Constraints:
(5) $\forall i \in \{1,\ldots,n\}$ alternative(JobMachine[i],all(w in ⟨i,j⟩: w.i==i) optionalJobMachine[w])
(6) $\forall i \in \{1,\ldots,n\}\ \forall k \in l(i)$ pulse(JobMachine[i],1) <= r_k
(7) $\forall j \in \{1,\ldots,m\}$ noOverlap(machines[j])

Figure A3. CP3: model based on tools as cumulative resources.

Appendix 2

Table B1. Average number of branches and fails for CP1 and CP1 with symmetry breaking models.

	CP1		CP1lex	
$\langle n, m, t \rangle$	Avg. # of Branches	Avg. # of Fails	Avg. # of Branches	Avg. # of Fails
⟨8, 2, 5⟩	1.50627268×10^8	5.2344061×10^7	1.54201084×10^8	4.7284024×10^7
⟨8, 2, 8⟩	1.16580514×10^8	4.1780104×10^7	1.22057892×10^8	3.8425479×10^7
⟨10, 2, 5⟩	1.72396274×10^8	6.0165895×10^7	1.68232832×10^8	5.5276052×10^7
⟨10, 2, 8⟩	1.33851179×10^8	4.6085622×10^7	1.26655325×10^8	4.1603443×10^7
⟨10, 3, 5⟩	1.71222474×10^8	6.7816670×10^7	1.69213050×10^8	6.2516160×10^7
⟨10, 3, 8⟩	1.54844735×10^8	6.2073556×10^7	1.64389490×10^8	6.0826295×10^7
⟨15, 2, 8⟩	1.50004372×10^8	5.5712641×10^7	1.42334153×10^8	4.5270937×10^7
⟨17, 2, 5⟩	1.56904914×10^8	5.6667851×10^7	1.49839828×10^8	5.0575140×10^7
⟨17, 2, 8⟩	1.42191050×10^8	5.1635796×10^7	1.50827109×10^8	4.8533845×10^7
⟨17, 3, 5⟩	1.58534086×10^8	6.1429137×10^7	1.57093320×10^8	5.7581480×10^7
⟨17, 3, 8⟩	1.55664665×10^8	6.2865508×10^7	1.51663664×10^8	5.7145331×10^7
⟨17, 4, 5⟩	1.54503390×10^8	6.4897927×10^7	1.51347451×10^8	6.1509759×10^7
⟨17, 4, 8⟩	1.55645572×10^8	6.6793592×10^7	1.49314356×10^8	6.2535244×10^7
⟨20, 2, 5⟩	1.47820144×10^8	5.4814411×10^7	1.48328233×10^8	5.0991199×10^7
⟨20, 2, 8⟩	1.43820294×10^8	5.4106801×10^7	1.47044387×10^8	4.8947480×10^7
⟨20, 3, 5⟩	1.55101524×10^8	6.1252912×10^7	1.49086316×10^8	5.7874399×10^7
⟨20, 3, 8⟩	1.54650697×10^8	6.3096654×10^7	1.49271182×10^8	6.0666189×10^7

Table B2. Average number of branches and fails for CP2 and CP2 with symmetry breaking models.

	CP2		CP2lex	
$\langle n, m, t \rangle$	Avg. # of Branches	Avg. # of Fails	Avg. # of Branches	Avg. # of Fails
⟨8, 2, 5⟩	2.2949923×10^7	1.1431125×10^7	4.4629874×10^7	2.2142969×10^7
⟨8, 2, 8⟩	2.9527773×10^7	1.4473197×10^7	6.8863751×10^7	3.3996334×10^7
⟨10, 2, 5⟩	5.199777×10^6	2.479515×10^6	1.1747743×10^7	5.606447×10^6
⟨10, 2, 8⟩	5.3454057×10^7	2.6028776×10^7	6.0359994×10^7	2.9591861×10^7
⟨10, 3, 5⟩	2.3796158×10^7	1.1541721×10^7	2.3435477×10^7	1.1380233×10^7
⟨10, 3, 8⟩	2.8223099×10^7	1.3514825×10^7	4.6201633×10^7	2.2560205×10^7
⟨15, 2, 8⟩	6.8728869×10^7	3.2494040×10^7	6.9634202×10^7	3.3015017×10^7
⟨17, 2, 5⟩	3.1349676×10^7	1.4596648×10^7	4.0615952×10^7	1.8947899×10^7
⟨17, 2, 8⟩	7.4533418×10^7	3.5057735×10^7	7.2859460×10^7	3.4351769×10^7
⟨17, 3, 5⟩	4.6043123×10^7	2.1470166×10^7	4.4251345×10^7	2.0647642×10^7
⟨17, 3, 8⟩	5.4461075×10^7	2.5599378×10^7	5.3329553×10^7	2.5075948×10^7
⟨17, 4, 5⟩	3.4000197×10^7	1.6101601×10^7	3.0312361×10^7	1.4354006×10^7
⟨17, 4, 8⟩	2.9782104×10^7	1.4278895×10^7	2.4739281×10^7	1.1845943×10^7
⟨20, 2, 5⟩	8.5261725×10^7	3.9321837×10^7	8.7257480×10^7	4.0288141×10^7
⟨20, 2, 8⟩	8.2348372×10^7	3.8477833×10^7	8.8242685×10^7	4.1313874×10^7
⟨20, 3, 5⟩	5.7072045×10^7	2.6492291×10^7	5.5275129×10^7	2.5692615×10^7
⟨20, 3, 8⟩	4.5616688×10^7	2.1542292×10^7	5.3901011×10^7	2.5401301×10^7

Table B3. Average number of branches and fails for the CP3 model.

	CP3	
$\langle n, m, t \rangle$	Avg. # of Branches	Avg. # of Fails
⟨8, 2, 5⟩	5.01317×10^5	2.27466×10^5
⟨8, 2, 8⟩	4.98668×10^5	2.23684×10^5
⟨10, 2, 5⟩	1.3104424×10^7	5.747677×10^6
⟨10, 2, 8⟩	1.426601×10^6	6.35127×10^5
⟨10, 3, 5⟩	1.5396509×10^7	6.993976×10^6
⟨10, 3, 8⟩	1.0974187×10^7	5.008304×10^6
⟨15, 2, 8⟩	8.5509783×10^7	3.6986239×10^7
⟨17, 2, 5⟩	5.8007317×10^7	2.4314092×10^7
⟨17, 2, 8⟩	8.6105271×10^7	4.2328961×10^7
⟨17, 3, 5⟩	5.5999797×10^7	2.4562085×10^7
⟨17, 3, 8⟩	6.9158267×10^7	3.0454357×10^7
⟨17, 4, 5⟩	4.0379357×10^7	1.8398663×10^7
⟨17, 4, 8⟩	2.2972111×10^7	1.0465313×10^7
⟨20, 2, 5⟩	1.09652462×10^8	4.5872966×10^7
⟨20, 2, 8⟩	1.10732200×10^8	4.7041585×10^7
⟨20, 3, 5⟩	7.0851281×10^7	3.0881577×10^7
⟨20, 3, 8⟩	7.1257959×10^7	3.1576153×10^7