Computers & Industrial Engineering 102 (2016) 160–165

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Flexible job shop scheduling problem with parallel batch processing machines: MIP and CP approaches

Andy M. Ham [a,*], Eray Cakici [b]

[a] Industrial Engineering, Liberty University, Lynchburg, VA, USA
[b] IBM, Istanbul, Turkey

ARTICLE INFO

Article history:
Received 14 July 2016
Received in revised form 1 November 2016
Accepted 2 November 2016
Available online 3 November 2016

Keywords:
FJSP
PBM
CP
MIP
Valid inequalities

ABSTRACT

A flexible job-shop scheduling problem (FJSP) with parallel batch processing machines (PBM) is addressed. First, we exhibit an enhanced mixed integer programming (MIP) model. Secondly, several valid inequalities are added for a reduction of solution space. Finally, we propose a constraint programming (CP) model, which is likely to be superior in many scheduling problems. Those different approaches are tested on a set of common problem instances from the literature. Computational results find three key lessons: the proposed MIP model significantly reduces computational time compared to the original model from the literature, the valid inequalities further reduce a computational time, and CP incomparably outperforms all three MIP models. Authors also discuss future adoption opportunities of CP, which has not got well deserved attention by OR/IE practitioners yet.

1. Introduction

The classical job shop scheduling problem (JSP) schedules a set of jobs on a set of machines with the objective to minimize a maximum completion time over all jobs (Cmax), subjected to the constraint that each job has an ordered set of operations, each of which must be processed on a predefined machine, whereas FJSP allows an operation to be processed on a machine out of a set of alternatives, which adds another dimension of complexity. The FJSP with parallel batch processing machine (PBM) inherits every complexity of the original FJSP. In addition, it has a set of parallel batch processing machines. The batching allows multiple jobs to be simultaneously processed as long as the total size of the batch does not exceed machine capacity. The processing time of a batch is dependent on the individual jobs in the batch, which is the maximum of individual processing times.

Considerable research has been devoted to FJSP in the literature. However, a literature review shows that there is not much work done about FJSP with PBM. The rest of this paper is organized as follows: a literature review is presented in Section 2 and the proposed MIP model and CP model are developed in Section 3. Computational results are reported in Section 4 and finally Section 5 covers the conclusion.

2. Previous related work

2.1. Flexible job shop scheduling problem

Researchers have addressed the FJSP mostly using heuristics. Despite the fact that those heuristics may generate fast and effective solutions, they are usually tailor-made. Moreover, the efficiency of these techniques strongly depends on the proper implementation and fine tuning of parameters since they combine the problem representation and the solution strategy into the same framework. In contrast, mathematical modelling approach divides the problem representation and the solution strategy (Kopanos, Méndez, & Puigjaner, 2010). Furthermore, as computer hardware and solvers have improved, practitioners have been able to formulate increasingly detailed and complex problems. Therefore, we explorer a mathematical modelling approach.

Table 1 shows an overview of FJSP mathematical models in the literature. The overview table originally created by Demir and İşleyen (2013) and is extended by Ham (in preparation). A vast number of researchers have addressed FJSP and its variants such as plan flexibility, setup, overlapping, preventive maintenance, etc. However, to the best of our knowledge, no published work has dealt with the FJSP with batching.

2.1.1. Constraint programming

In spite of its success for effectively solving many scheduling problems (Darby-Dowman, Little, Mitra, & Zaffalon, 1997; Öztürk,

* Corresponding author.
E-mail addresses: mham@liberty.edu (A.M. Ham), eray.cakici@gmail.com (E. Cakici).

http://dx.doi.org/10.1016/j.cie.2016.11.001
0360-8352/

Table 1
An overview of FJSP mathematical models.

References	Highlights	Journal
Liu and MacCarty (1997)	Sequence dependent setup times	EJOR
Kim and Egbelu (1999)	Process plan flexibility	IJPR
Thomalla (2001)	Alternative process plan	IJPE
Low and Wu (2001)	Sequence dependent setup times	IJPR
Tamaki, Ono, Murao, and Kitamura (2001)	Sequence dependent setup times	IEEE
Lee, Jeong, and Moon (2002)	Process plan flexibility	C&IE
Torabi, Karimi, and Fatemi Ghomi (2005)	Homogenous machines	IJPE
Imanipour (2006)	Sequence dependent setup times	IEEE
Low, Yip, and Wu (2006)	Sequence independent setup times	C&OR
Gao, Gen, and Sun (2006)	Flexible preventive maintenance	JIM
Fattahi et al. (2007)	–	JIM
Mehrabad and Fattahi (2007)	Sequence dependent setup times	IJAMT
Zhang, Shao, Li, and Gao (2009)	–	C&IE
Lin and Jia-zhen (2009)	Sequence independent setup times	SETP
Fattahi, Jolai, and Arkat (2009)	Overlapping	AMM
Özgüven, Özbakır, and Yavuz (2010)	Process plan flexibility	AMM
Fattahi and Fallahi (2010)	Disturbances	JMST
Ham, Lee, and Kim (2011)	–	IJPR
Moradi, Fatemi Ghomi, and Zandieh (2011)	Preventive maintenance	ESA
Zhang, Manier, and Manier (2012)	Transportation constraints	C&OR
Demir and İşleyen (2013)	Evaluation of MIP models	AMM
Jalilvand-Nejad and Fattahi (2015)	Sequence dependent setup times	JIM
Ham (in preparation)	With batching	In Review

Tunalı, Hnich, & Örnek, 2013), CP has not got well deserved attention by OR/IE practitioners yet. We can list up two reasons: First, the syntax and keywords provided by a commercial CP package look unfamiliar to the eye of practitioners who are used to MIP formulation. For instance, there is a special variable called *interval* in CP Optimizer, the CP engine available in the IBM ILOG CPLEX Optimization Studio. The interval variable can represent optional tasks to be scheduled. The domain of an interval variable x is a subset of $\{Absent\} \cup \{[s,e)|s,e \in \mathbb{Z}, s \leqslant e\}$. As any decision variable in an optimization problem, an interval variable x is said to be fixed if its domain is reduced to a singleton. When an interval is present, s represents the start time, and e the end time. Therefore, one interval variable in CP replaces several decision variables of MIP (typically start and end variables and a Boolean variable representing the presence status). While it definitely provides practitioners an efficient environment, it takes time to understand a CP formulation. Another reason is that there is no standard in CP formulation. Namely, it varies to each CP packages, whereas we have a very similar MIP formulation across packages. As a practitioner, we want to see a standardized CP formulation.

On the other hand, there are many reasons why CP would be soon widely adopted by OR/IE practitioners. Firstly, CP's natural formulation is closer to the problem description than restricted linear programming formulation. This is specifically true to scheduling problem which deals with a time and precedence relation. Practitioners would want to avoid the headache of writing a complex linear programming formulation. For instance, *dvar sequence* in <intervalName> prescribes there must be a sequence among the interval variables. This construct replaces a tedious pair of disjunctive constraints in many MIP models. Secondly, a concise CP code provides a flexibility and scalability to practitioners. CP can address non-linear constraints by nature while being capable of solving large problems. Also, today's competitive market requires more flexible systems that respond rapidly to changes in the market condition. The complex MIP and heuristic codes seem to not fit well in this dynamic environment. Thirdly, metaheuristics are tailor-made, requiring a fine tuning of parameters to reach at the best performance, whereas CP belongs to a category of off-the-rack which does not much require the fine tuning. Practitioners simply provide a description of the problem. All settings of search algorithms and detailed tunings are automatically done by CP engine. Let us quote other literatures. Barták, Salido, and Rossi (2010) points out that CP is a combination of defining constraints about the problem with variables and finding a solution that satisfies all the constraints, similar to MIP. However, in CP, constraints are used actively to infer new constraints and to reduce domains of variables by removing certain values that violate constraints. We now borrow the classical "crypto-arithmetic" puzzle explained in Apt (1999) and Barták et al. (2010) to understand the inference. The distinct variables {S,E,N,D,M,O,R,Y} represent digits between 0 and 9 and the task is finding values for them such that the following arithmetic operation is correct:

$$\begin{array}{r} \text{S E N D} \\ +\text{M O R E} \\ \hline \text{M O N E Y} \end{array}$$

The constraints are

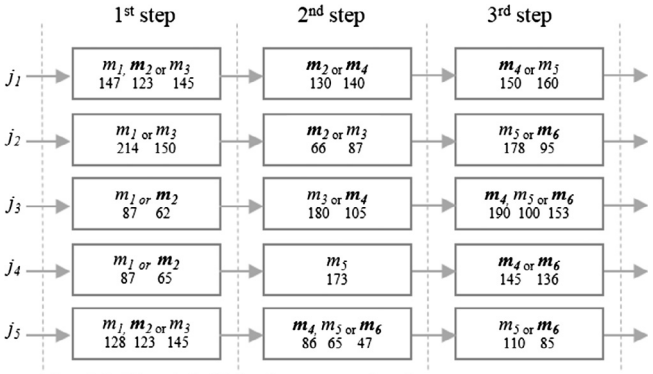
$$\begin{aligned} &10^3(S+M)+10^2(E+O)+10(N+R)+D+E \\ &\quad = 10^4M+10^3O+10^2N+10E+Y; \\ &S,E,N,D,M,O,R,Y \in \{0,\ldots,9\}, \quad \textit{All different}\{S,E,N,D,M,O,R,Y\}; \end{aligned}$$

By using an inference (or constraint propagation), the search space can be reduced to the following.

$$\begin{aligned} &S \in \{9\}, \quad E \in \{4,\ldots,7\}, \quad N \in \{5,\ldots,8\}, \quad D \in \{2,\ldots,8\}, \\ &M \in \{1\}, \quad O \in \{0\}, \quad R \in \{2,\ldots,8\}, \quad Y \in \{2,\ldots,8\} \end{aligned}$$

Note the size of permutations are reduced from 10^7 to 5488 owing to the inferences.

In this study, we explore this very promising approach, CP, which is likely to be superior in many scheduling problems. Furthermore, we enhance MIP formulation compared to the one in the literature.



Note: Machines in bold font have a capacity of two.
Numbers under each machine represent a processing time at steps.
Batching requirement is added to original instance MFJS1 [13],[2]

Fig. 1. FJSP instance of 5-job, 6-machine, and 3-step with batching.

3. Formulations

We assume all machines and jobs are available at time 0. We also assume that batch processing machine can process different products simultaneously (compatible job families). Fig. 1 represents a FJSP with batching instance of 5-job, 6-machine, and 3-step (Ham, in preparation).

The notation used in this mathematical modelling is summarized in the following:

Sets	
J	jobs (j)
S	steps (s)
M	machines (m)
B	batches (b)
Parameters	
$par_{j,s,m}^{ptime}$	processing time
par_{m}^{capa}	capacity of machine m
nj	number of jobs
ns	number of steps
L	$\sum_s MAX_{j,m}\{par_{j,s,m}^{ptime}\}$
Decision variables	
X_{jsb}	1 if job j is assigned to batch b at step s; 0
Y_{bsm}	1 if batch b is assigned to machine m at step s; 0 otherwise
Resultant variables	
$J_{j,s}^{start}$	start time of job j at step s
$J_{j,s}^{complete}$	completion time of job j at step s
$M_{b,s}^{start}$	start time of batch b at step s
$M_{b,s}^{complete}$	completion time of batch b at step s
$M_{b,s}^{ptime}$	processing time of batch b at step s
$Cmax$	maximum completion time (makespan)

3.1. Mixed integer programming

Routing

$$\sum_b X_{jsb} = 1 \quad \forall j,s \tag{1.1}$$

$$Nj\left(\sum_m Y_{bsm}\right) \geqslant \sum_j X_{jsb} \quad \forall j,b \tag{1.2}$$

$$\sum_m Y_{bsm} \leqslant 1 \quad \forall j,b \tag{1.3}$$

$$\sum_j X_{jsb} \leqslant par_m^{capa} + nj(1 - Y_{bsm}) \quad \forall b,s,m \tag{1.4}$$

$$X_{jsb} + Y_{bsm} \leqslant 1 \quad \forall j,b,s,m : par_{j,s,m}^{ptime} = 0 \tag{1.5}$$

Scheduling

$$M_{b,s}^{ptime} \geqslant par_{j,s,m}^{ptime} - (2 - X_{jsb} - Y_{bsm})L \quad \forall j,b,s,m : s = 1 \tag{1.6}$$

$$M_{b,s}^{complete} \geqslant J_{j,s-1}^{complete} + M_{b,s}^{ptime} - (2 - X_{jsb} - Y_{bsm})L \quad \forall j,b,s,m : s \geqslant 2 \tag{1.7}$$

$$M_{b,s}^{complete} \leqslant J_{j,s}^{complete} - (1 - X_{jsb})L \quad \forall j,b,s \tag{1.8}$$

$$M_{b,s}^{complete} \geqslant J_{j,s}^{complete} - (1 - X_{jsb})L \quad \forall j,b,s \tag{1.9}$$

$$M_{b,s}^{complete} - M_{b,s}^{ptime} = M_{b,s}^{start} \quad \forall b,s \tag{1.10}$$

$$M_{b,s}^{complete} \leqslant M_{\hat{b},s}^{start} + (2 - Y_{bsm} - Y_{\hat{b}sm})L \quad \forall s,m : b \geqslant \hat{b} + 1 \tag{1.11}$$

$$M_{b,s}^{complete} \leqslant M_{b,s+1}^{start} + (2 - Y_{bsm} - Y_{\hat{b},s+1,m})L \quad \forall s,m : s \leqslant ns - 1 \tag{1.12}$$

Measuring

$$Cmax \geqslant M_{b,s}^{complete} \quad \forall b,s \tag{1.13}$$

$$Minimize\ Cmax \tag{1.14}$$

To our best knowledge, only MIP formulation in literature is provided by Ham (in preparation) for FJSP with batching. Its mathematical model is provided in Appendix. The size of a linear program is typically given by the number of constraints and the number of variables. A large number of variables and constraints increases the computational time which a linear program is solved. In order to reduce number of variables and constraints in 4-indexed formulation of the paper, here we propose a 3-indexed formulation. A set of batches B as equal to |J| is introduced for each stage since the largest value for the number of batches at most can be as equal to the number of jobs. The batches are numbered in ascending order of their starting times. Therefore, sequencing of batches are handled without defining an additional variable index.

Constraints (1.1)–(1.5) handle routing. Constraint (1.1) assigns every job to a batch in each stage, Constraint (1.2) ensures that if a job is assigned to a batch then that batch should be assigned to a machine, batches cannot be assigned more than one machine in Constraint (1.3) and batch capacity cannot be exceeded in Constraint (1.4). Finally, Constraint (1.5) ensures that if a job cannot be processed by a machine then batch including that job cannot be processed on that machine as well. Constraints (1.6)–(1.12) define a scheduling where L is a big number. Constraint (1.6) defines the processing time of a batch on a machine, which is represented by the longest time of all jobs in the batch. Constraints (1.7)–(1.9) determine the completion time of a batch. Constraint (1.10) calculates the start time of a batch. Constraint (1.11) ensures a subsequent batch cannot start until a preceding batch completes, when the batches visit a same machine. Similarly, Constraint (1.12) ensures a batch cannot start until the batch completes at a preceding step. Finally, Constraint (1.13) determines the makespan and Objective (1.14) minimizes it.

3.2. Mixed integer programming with valid inequalities

In order to strengthen the formulation by generating better lower bounds, constraints (1.15)–(1.17) are added into the model as additional valid inequalities. These constraints are valid for feasible regions of the problem studied and can reduce solution time basically by tightening the feasible region of the LP relaxation. Constraint (1.15) ensures that the start time of a job at a stage is at least the total minimum processing times of that job at earlier stages.

$$J_{j,s}^{start} \geqslant \sum_{k<s:k\in Batches} MIN_{m:par_{j,s,m}^{ptime}>0}\left(par_{j,s,m}^{ptime}\right) \quad \forall j,s \tag{1.15}$$

Constraints (1.16) and (1.17) are linking the start time of batches with jobs assigned to them.

$$M_{b,s}^{start} \leqslant J_{j,s}^{start} + (1 - X_{jsb})L \quad \forall j,b,s \tag{1.16}$$

$$M_{b,s}^{start} \geqslant J_{j,s}^{start} - (1 - X_{jsb})L \quad \forall j,b,s \tag{1.17}$$

3.3. Constraint programming

Here we represent the problem in CP by using IBM ILOG CP Optimizer. The details of the modelling language will not be given. We recommend readers to refer to (IBM Software, 2010, 2012; Laborie & Rogerie, 2008; Laborie, Rogerie, Shaw, & Vilim, 2009).

Parameters	
$Oper_{opId,j,s}$	list of operation $O_{j,s}$. We refer the combination of job and step as an operation.
$O2M_{opId,m,(pt)}$	list of eligible machines of $O_{j,s}$ and their processing times.
Decision variables	
Itv^{Oper}	*interval* object of $Oper_{opId,j,s}$: an *interval* variable represents an interval of time characterized by a start time, an end time, and a size.
Itv^{O2M}, *optional*	*interval* object of $O2M_{opId,m,pt}$: this *interval* is declared *optional* to model the parallel machines.
Functions	
$State_m^{Batch}$	*state* of each machine representing a batch. The intervals of the *state function* represent the different batches. The treatment of each item is modelled by an *interval* variable. Jobs serviced in the same batch need to be synchronized; they have the same start and end time.
$Cumul_m^{Load} \sum_{O_{j,s}} pulse(Itv^{O2M}, 1)$	Each assignment of operation to machine increases the *cumul function* at the start of the usage of a machine, and decreases the function when the machine is released at its end time.

The following is a CP representation of the problem under study.

Routing

$$Alternative(Itv^{Oper}, Itv^{O2M}) : Itv^{Oper} \rightarrow opId = Itv^{O2M} \quad (2.1)$$

Sequencing

$$endBeforeStart\left(Itv_{j,s}^{Oper}, Itv_{j,\hat{s}}^{Oper}\right) : O_{j,s} \text{ has to precede } O_{j,\hat{s}} \quad (2.2)$$

Synchronizing

$$alwaysEqual\left(State_m^{Batch}, Itv^{O2M}\right) : State_m^{Batch} \rightarrow m = Itv^{O2M} \rightarrow m \quad (2.3)$$

Batching

$$Cumul_m^{Load} \leqslant par_m^{capa} \quad (2.4)$$

Measuring

$$minimize\ max\{endOf(Itv^{Oper})\} \quad (2.5)$$

Constraint (2.1) forces to assign each operation to exactly one of machines. Both intervals of Itv^{Oper} and Itv^{O2M} start and end together with this chosen one. Constraint (2.2) ensures the precedence relation between the operations associated with each job. Constraint (2.3) specifies that Itv^{O2M} in the same batch (state) is synchronized. Constraint (2.4) ensures that the total job size of a batch cannot exceed the machine capacity. Finally, Objective (2.5) minimizes the makespan. The specialized keywords and constructs of CP Optimizer allow us to represent the complex scheduling problem in a concise code.

4. Computational study

In this section, we test the effectiveness of our proposed models. MIP and CP models are generated by IBM OPL 12.6.3 on a personal computer with an Intel Core i5-3470 @ 3.2 Ghz processor and 16 GB RAM. We use the same test problems instances used by Fattahi, Mehrebad, and Jolai (2007). They randomly generated a total of 20 FJSP instances. The instances are divided to two categories: small size problems (SFJS1:10) and medium and large size problems (MFJS1:10). The problem instances are determined by size of the problem ($n/h/m$) in which index n denotes number of jobs, h denotes the maximum number of operations for all jobs and m denotes the machine number.

The instances however do not have a batching requirement so we simply assume even (odd) numbered machines have a capacity of two (one). The customized CPLEX logs have the scheduling results. All the test instances, log files, and results are located at https://dl.dropboxusercontent.com/u/57440903/FJSP_Batching_CP.zip.

Table 2 summarizes the computational results of the small size problems. Columns 1–2 show the name of instances used by Fattahi et al. (2007) and size, respectively. Columns 3–5 contain the best solutions generated by Ham (in preparation) which are reported within a 300 s. The CPU indicates a computational runtime (in seconds) and the BIN shows the number of binary integer variables. Columns 6–8 (9–11) contain the best solutions generated by the proposed MIP models: without (with) valid inequalities respectively. Finally, Columns 12–13 report the best solutions generated by the proposed CP. All four models find an optimal solution within 300 s except SFJS10 instance by Ham. The proposed MIP model dramatically reduces the computational time when it compares to the one from Ham. The valid inequalities further reduce a computational time by 49% compared to the one without valid inequalities. CP incomparably outperforms all three MIP models.

Similarly, Table 3 summarizes the computational results of the medium/large size problems. All three MIP models are unable to find an optimal solution within 300 s except MFJS6 instance by MIP with valid inequalities, whereas CP model still finds an optimal solution for all instances in 3.72 s on average. This experimentation confirms CP's outstanding performance for the problem under study.

5. Conclusion and future works

A flexible job-shop scheduling problem (FJSP) with parallel batch processing machine (PBM) is visited. We exhibit an enhanced mixed integer programming (MIP) model, add several valid inequalities for a reduction of solution space, and propose a constraint programming (CP) model. Those different approaches are tested on a set of common problem instances from the literature. Computational results find three key lessons: the proposed MIP model significantly reduces a computational time compared to the original model from the literature, the valid inequalities further improve a computational time by tightening the formulation, and CP incomparably outperforms all three MIP models.

In addition, authors discuss the adoption opportunities of CP which has not got well deserved attention by OR/IE practitioners

Table 2
Comparison of four different models with the small size problems.

Problem	Size	Ham (in preparation)			MIP			MIP (valid inequalities)			CP	
		Cmax	cpu	bin	*Cmax*	cpu	bin	*Cmax*	cpu	bin	*Cmax*	cpu
SFJS1	2/2/2	**66**	0.05	32	**66**	0.01	12	**66**	0.02	12	**66**	0.08
SFJS2	2/2/2	**107**	0.05	24	**107**	0.01	9	**107**	0.02	9	**107**	0.00
SFJS3	3/2/2	**208**	0.31	60	**208**	0.05	30	**208**	0.06	30	**208**	0.02
SFJS4	3/2/2	**272**	0.06	60	**272**	0.02	30	**272**	0.01	30	**272**	0.00
SFJS5	3/2/2	**100**	0.14	72	**100**	0.06	30	**100**	0.06	30	**100**	0.00
SFJS6	3/3/2	**320**	12.29	135	**320**	0.31	48	**320**	0.36	48	**320**	0.00
SFJS7	3/3/5	**397**	9.77	162	**397**	0.64	54	**397**	0.05	54	**397**	0.00
SFJS8	3/3/4	**216**	5.34	162	**216**	0.51	51	**216**	0.08	51	**216**	0.00
SFJS9	3/3/3	**210**	2.95	162	**210**	0.55	51	**210**	0.06	51	**210**	0.00
SFJS10	4/3/5	**516**	300	240	**516**	5.9	84	**516**	3.38	84	**516**	0.01
Average		241	33.10	111	241	0.81	40	241	0.41	40	241	0.01

Optimal values in bold.

Table 3
Comparison of four different models with the medium/large size problems.

Problem	Size	Ham (in preparation)			MIP			MIP (valid inequalities)			CP	
		Cmax	cpu	bin	*Cmax*	cpu	bin	*Cmax*	cpu	bin	*Cmax*	cpu
MFJS1	5/3/6	410	300	495	410	300	130	410	300	130	**410**	0.01
MFJS2	5/3/7	410	300	585	410	300	140	410	300	140	**410**	0.01
MFJS3	6/3/7	420	300	864	420	300	192	420	300	192	**420**	0.01
MFJS4	7/3/7	506	300	1176	503	300	238	506	300	238	**503**	0.04
MFJS5	7/3/7	488	300	1155	488	300	245	488	300	245	**488**	0.03
MFJS6	8/3/7	631	300	1488	614	300	312	**614**	82.84	312	**614**	0.05
MFJS7	8/4/7	916	300	2496	804	300	408	848	300	408	**789**	0.67
MFJS8	9/4/8	896	300	3096	842	300	504	804	300	504	**774**	0.88
MFJS9	11/4/8	nf	300	4532	925	300	704	876	300	704	**843**	24.27
MFJS10	12/4/8	3418	300	5376	1119	300	816	1105	300	816	**985**	11.19
Average		899	300.0	2126	654	300.0	369	648	278.3	369	624	3.72

Optimal values in bold.

yet. Firstly, the syntax and keywords provided by a commercial CP package look unfamiliar to the eye of practitioners who used to MIP formulation. Another reason is that there is no standard in CP formulation. Namely, it varies to each CP packages, whereas we have a very similar MIP formulation across packages. Authors also provide the reasons why CP would be soon welcomed by the practitioners. Firstly, CP's natural formulation is closer to the problem description than the restricted linear programming formulation. Secondly, a concise CP code provides a flexibility and scalability to practitioners. Thirdly, unlike metaheuristics which are tailor-made requiring a fine tuning of parameters to reach at the best performance, CP is off-the-rack. Namely, practitioners provide a high level description of the problem only. All settings of search algorithms and detailed tunings are automatically done by CP engine. Finally, CP outperforms MIP in the scheduling problems as we demonstrate in this study.

CP is a form of adaptive search and with rapidly increasing interest in artificial intelligence and cognitive systems so that one can expect significant performance improvements in CP engines. Therefore, it is worth to continue exploring different CP-based approaches for this problem *i.e.* a hybrid approach (MIP-CP and/or Heuristic-CP) can be analyzed in future research. Alternatively, we can enhance the proposed MIP model and come up with more efficient one. This research can also be further extended to application side. In semiconductor manufacturing, the wet-diffusion areas process 2–4 consecutive steps with parallel batching machines. Another application is to schedule jobs having a time constraints between consecutive steps (Klemmt & Mönch, 2012; Sadeghi, Dauzere-Peres, Yugma, & Lepelletier, 2015; Sun, Choung, Lee, & Jang, 2005). Both problems can be represented as FJSP with batching.

Appendix A

Routing

$$\sum_{m,k} X_{jsmk} = 1 \quad \forall j,s \tag{A.1}$$

$$\sum_{j,s} (X_{jsmk}) \leqslant par_m^{capa} \quad \forall k,m \tag{A.2}$$

Scheduling

$$M_{m,k}^{ptime} \geqslant \left(par_{j,s,m}^{ptime}\right) X_{jsmk} \quad \forall j,s,m,k \tag{A.3}$$

$$J_{j,1}^{arrival} = par_j^{release} \quad \forall j \tag{A.4}$$

$$M_{m,k}^{start} \geqslant J_{j,s}^{arrival} + M(X_{jsmk} - 1) \quad \forall j,s,m,k \tag{A.5}$$

$$M_{m,k}^{complete} = M_{m,k}^{start} + M_{m,k}^{ptime} \quad \forall m,k \tag{A.6}$$

$$J_{j,s+1}^{arrival} \geqslant M_{m,k}^{complete} + M(X_{jsmk} - 1) \quad \forall j,m,k : s < |S| \tag{A.7}$$

$$M_{m,k+1}^{start} \geqslant M_{m,k}^{complete} \quad \forall m : k < |K| \tag{A.8}$$

Measuring

$$Cmax \geqslant M_{m,k}^{complete} + M(X_{jsmk} - 1) \quad \forall j,s,m,k \tag{A.9}$$

$$Minimize\ Cmax \tag{A.10}$$

Constraint (A.1) ensures that jobs are assigned to one of the available slots. Machine capacity is taken into consideration in Constraint (A.2). Then, Constraint (A.3) defines the processing time of a batch on a machine, which is represented by the longest time of all jobs in the batch. Constraint (A.4) considers the release time of a job. Constraint (A.5) ensures that a batch cannot start its processing until all jobs assigned to the corresponding batch become ready. Constraint (A.6) determines the completion time of a batch. Constraint (A.7) ensures that the available time of a job is greater than or equal to the completion time at the very previous step. Constraint (A.8) ensures the precedence relationship between batches at the same machine. Finally, Constraint (A.9) determines the makespan and Objective (A.10) minimizes it.

References

Apt, K. (1999). *Principles of constraint programming*. Cambridge University Press.

Barták, R., Salido, M. A., & Rossi, F. (2010). Constraint satisfaction techniques in planning and scheduling. *Journal of Intelligent Manufacturing, 21*(1), 5–15.

Darby-Dowman, K., Little, J., Mitra, G., & Zaffalon, M. (1997). Constraint logic programming and integer programming approaches and their collaboration in solving an assignment scheduling problem. *Constraints, 1*(3), 245–264.

Demir, Y., & İşleyen, S. K. (2013). Evaluation of mathematical models for flexible job-shop scheduling problems. *Applied Mathematical Modelling, 37*(3), 977–988.

Fattahi, P., & Fallahi, A. (2010). Dynamic scheduling in flexible job shop systems by considering simultaneously efficiency and stability. *CIRP Journal of Manufacturing Science and Technology, 2*, 114–123.

Fattahi, P., Jolai, F., & Arkat, J. (2009). Flexible job shop scheduling with overlapping in operations. *Applied Mathematical Modelling, 33*, 3076–3087.

Fattahi, P., Mehrebad, M. S., & Jolai, F. (2007). Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. *Journal of Intelligent Manufacturing, 18*, 331–342.

Gao, J., Gen, M., & Sun, L. (2006). Scheduling jobs and maintenances in flexible job shop with a hybrid genetic algorithm. *Journal of Intelligent Manufacturing, 17*, 493–507.

Ham, A. (2016). Flexible job shop scheduling problem with parallel batch processing machine for compatible job families (in preparation).

Ham, M., Lee, Y. H., & Kim, S. H. (2011). Real-time scheduling of multi-stage flexible job shop floor. *International Journal of Production Research, 49*(12), 3715–3730.

IBM Software (2010). Modeling with IBM ILOG CPLEX CP optimizer – practical scheduling examples [White paper]. Retrieved from <http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio/practical-scheduling-examples/>.

IBM Software (2012). IBM ILOG CPLEX Optimization Studio V12.5.1.

Imanipour, N. (2006). Modeling & solving flexible job shop problem with sequence dependent setup times. *Proceedings of the international conference on service systems and service management, October 25–27, 2006* (Vol. 2, pp. 1205–1210). IEEE.

Jalilvand-Nejad, A., & Fattahi, P. (2015). A mathematical model and genetic algorithm to cyclic flexible job shop scheduling problem. *Journal of Intelligent Manufacturing, 26*(6), 1085–1098.

Kim, K.-H., & Egbelu, P. J. (1999). Scheduling in a production environment with multiple process plans per job. *International Journal of Production Research, 37*, 2725–2753.

Klemmt, A., & Mönch, L. (2012). Scheduling jobs with time constraints between consecutive process steps in semiconductor manufacturing. In *Proceedings of the winter simulation conference. Winter simulation conference* (pp. 194).

Kopanos, G. M., Méndez, C. A., & Puigjaner, L. (2010). MIP-based decomposition strategies for large-scale scheduling problems in multiproduct multistage batch plants: A benchmark scheduling problem of the pharmaceutical industry. *European Journal of Operational Research, 207*(2), 644–655.

Laborie, P., & Rogerie, J. (2008). Reasoning with conditional time-intervals. In *FLAIRS conference* (pp. 555–560).

Laborie, P., Rogerie, J., Shaw, P., & Vilim, P. (2009). Reasoning with conditional time-intervals. Part II: An algebraical model for resources. In *FLAIRS conference* (pp. 201–206).

Lee, Y. H., Jeong, C. S., & Moon, C. (2002). Advanced planning and scheduling with outsourcing in manufacturing supply chain. *Computers & Industrial Engineering, 43*, 351–374.

Lin, L., & Jia-zhen, H. (2009). Multi-objective flexible job-shop scheduling problem in steel tubes production. *Systems Engineering Theory Practice, 29*(8), 117–126.

Liu, J., & MacCarty, B. L. (1997). A global milp model for FMS scheduling. *European Journal of Operational Research, 100*, 441–453.

Low, C. Y., & Wu, T. H. (2001). Mathematical modelling and heuristic approaches to operation scheduling problems in an FMS environment. *International Journal of Production Research, 39*, 689–708.

Low, C., Yip, Y., & Wu, T. H. (2006). Modeling and heuristics of FMS scheduling with multiple objectives. *Computers & Operations Research, 33*, 674–694.

Mehrabad, M. S., & Fattahi, P. (2007). Flexible job shop scheduling with tabu search algorithms. *International Journal of Advanced Manufacturing Technology, 32*, 563–570.

Moradi, E., Fatemi Ghomi, S. M. T., & Zandieh, M. (2011). Bi-objective optimization research on integrated fixed time interval preventive maintenance and production for scheduling flexible job-shop problem. *Expert Systems with Applications, 38*, 7169–7178.

Özgüven, C., Özbakır, L., & Yavuz, Y. (2010). Mathematical models for job-shop scheduling problems with routing and process plan flexibility. *Applied Mathematical Modelling, 34*, 1539–1548.

Öztürk, C., Tunalı, S., Hnich, B., & Örnek, M. A. (2013). Balancing and scheduling of flexible mixed model assembly lines. *Constraints, 18*(3), 434–469.

Sadeghi, R., Dauzere-Peres, S., Yugma, C., & Lepelletier, G. (2015). Production control in semiconductor manufacturing with time constraints. In *Advanced semiconductor manufacturing conference (ASMC), 2015 26th annual SEMI* (pp. 29–33). IEEE.

Sun, D. S., Choung, Y. I., Lee, Y. J., & Jang, Y. C. (2005). Scheduling and control for time-constrained processes in semiconductor manufacturing. In *Semiconductor manufacturing, 2005. ISSM 2005, IEEE international symposium on* (pp. 295–298). IEEE.

Tamaki, H., Ono, T., Murao, H., & Kitamura, S. (2001). Modeling and genetic solution of a class of flexible job shop scheduling problems. *Proceedings of the IEEE symposium on emerging technologies and factory automation* (Vol. 2, pp. 343–350). IEEE.

Thomalla, C. S. (2001). Job shop scheduling with alternative process plans. *International Journal of Production Economics, 71*, 125–134.

Torabi, S. A., Karimi, B., & Fatemi Ghomi, S. M. T. (2005). The common cycle economic lot scheduling in flexible job shops: The finite horizon case. *International Journal of Production Economics, 97*, 52–65.

Zhang, Q., Manier, H., & Manier, M.-A. (2012). A genetic algorithm with tabu search procedure for flexible job shop scheduling with transportation constraints and bounded processing times. *Computers & Operations Research, 39*, 1713–1723.

Zhang, G., Shao, X., Li, P., & Gao, L. (2009). An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. *Computers & Industrial Engineering, 56*, 1309–1318.