



Job shop scheduling with a combination of four buffering constraints

Shi Qiang Liu, Erhan Kozan, Mahmoud Masoud, Yu Zhang & Felix T.S. Chan

To cite this article: Shi Qiang Liu, Erhan Kozan, Mahmoud Masoud, Yu Zhang & Felix T.S. Chan (2018) Job shop scheduling with a combination of four buffering constraints, International Journal of Production Research, 56:9, 3274-3293, DOI: [10.1080/00207543.2017.1401240](https://doi.org/10.1080/00207543.2017.1401240)

To link to this article: <https://doi.org/10.1080/00207543.2017.1401240>



Published online: 17 Nov 2017.



Submit your article to this journal [↗](#)



Article views: 575



View related articles [↗](#)




View Crossmark data [↗](#)



Citing articles: 11 View citing articles [↗](#)

Job shop scheduling with a combination of four buffering constraints

Shi Qiang Liu^a, Erhan Kozan^b , Mahmoud Masoud^b, Yu Zhang^c and Felix T.S. Chan^{d*}

^aSchool of Economics and Management, Fuzhou University, Fuzhou, China; ^bSchool of Mathematical Sciences, Queensland University of Technology, Brisbane, Australia; ^cCivil and Environmental Engineering, University of South Florida, Tampa, FL, USA; ^dDepartment of Industrial & Systems Engineering, Hong Kong Polytechnic University, Hong Kong

(Received 11 April 2017; accepted 19 October 2017)

In this paper, a new scheduling problem is investigated in order to optimise a more generalised Job Shop Scheduling system with a Combination of four Buffering constraints (i.e. *no-wait*, *no-buffer*, *limited-buffer* and *infinite-buffer*) called *CBJSS*. In practice, the *CBJSS* is significant in modelling and analysing many real-world scheduling systems in chemical, food, manufacturing, railway, health care and aviation industries. Critical problem properties are thoroughly analysed in terms of the Gantt charts. Based on these properties, an applicable mixed integer programming model is formulated and an efficient heuristic algorithm is developed. Computational experiments show that the proposed heuristic algorithm is satisfactory for solving the *CBJSS* in real time.

Keywords: job shop scheduling; buffer management; blocking; no-wait; mixed integer programming; constructive algorithm; best-insertion-heuristic algorithm

1. Introduction

The classical job shop scheduling (*JSS*) problem is regarded as one of the most difficult problems in combinatorial optimisation. An indication of its difficulty was given by the fact that a 10-job, 10-machine instance formulated for the first time (Fisher and Thompson 1963) was exactly solved after over 30 years by a branch-and-bound algorithm with more than 5-h running time (Carlier and Pinson 1989). Because of its significance from a theoretic view, the *JSS* is still an important research topic till now (Amirghasemi and Zamani 2015; Peng, Lü, and Cheng 2015; Zhang and Chiong 2016; Zhang et al., *forthcoming*).

However, the capacity of intermediate buffer storage in a classical job shop is assumed as infinite, which implies that specific buffering constraints (i.e. *no-wait*, *no-buffer* and *limited-buffer*) are neglected. In practice, this assumption results in inapplicability for modelling many industrial systems. For example, in the food industry, the canning operation must immediately follow the cooking operation to ensure freshness, which means that the *no-wait* constraint occurs in this situation (Hall and Sriskandarajah 1996). For instance, the *no-buffer* (*blocking*) constraint is required in chemical industry, where partially processed chemical products sometimes have to be temporarily kept in the processing machine because of high temperature or safety issues (Pacciarelli 2002). In manufacturing industry, the *limited-buffer* constraint is critical to alleviate abrupt changes in fabrication lines, as intermediate buffers can accommodate the product parts after a processing equipment unit or supply them to the next equipment unit among the contiguous process steps (Toba 2005; Chan and Choy 2011).

In some industrial systems, *no-wait*, *no-buffer*, *limited-buffer* constraints must be considered in an integrated way (Liu and Kozan 2009a). Investigation of a job shop system with combined buffering constraints is beneficial to analyse and optimise the operations in several industries. One implementation arises in healthcare industry, in which both outpatients and inpatients are serviced in the hospital facilities. For example, the *no-wait* constraint incurs when an urgent surgical case is treated for an acute outpatient; while the *blocking* constraint happens when an inpatient has to remain in a ward bed until an operating theatre becomes available (Chien, Tseng, and Chen 2008; Pham and Klinkert 2008; Ruan et al. 2016). Another important implementation occurs in railway industry due to the lack of crossing loops and the consideration of train's priority (D'Ariano, Pacciarelli, and Pranzo 2007, 2008; Burdett and Kozan 2009, 2010; Liu and Kozan 2009b, 2011a, 2011b; Masoud, Kozan, and Kent 2011, 2015; Bürgy and Gröflin 2016; Masoud, Kent et al. 2016; Masoud, Kozan et al. 2016; Masoud et al. 2017). In a railway network, a tunnel section has to impose the *no-wait* constraint for safety; a bridge section may require the *no-buffer* constraint due to absence of crossing loops; a loading

*Corresponding author. Email: f.chan@polyu.edu.hk

section with sidings must consider the *limited-buffer* constraint; and a depot section could allow the *infinite-buffer* constraint. One recent implementation of job shop scheduling with buffering constraints comes from aviation management, which aims to optimise the take-off and landing operations at a busy European airport terminal with limited-capacity infrastructure (Samà, D'Ariano et al., 2017).

In the literature about multi-stage scheduling with buffering constraints, the initial research efforts dealt with the flow shop scheduling (*FSS*) problem that is the simplified version of the *JSS* problem. To better understand the difference between the *JSS* and the *FSS*, the key characteristics of the classical *FSS* and *JSS* problems are stated as follows (Liu and Ong 2002, 2004; Liu, Ong, and Ng 2005; Liu and Kozan 2012a; Bai et al. 2017; Rossit, Tohmé, and Frutos, forthcoming). Given a set of jobs that have to be processed on a set of machines, each job consists of multiple operations. In the *FSS*, the operations of every job are required to be processed on such a set of machines in the same unidirectional order. In comparison, in the *JSS*, each job has a prescribed processing order through the machines, but the processing order for each job may be different. For the *FSS* with buffering constraints, the following papers published in the leading journals are referred. Leisten (1990) presented the initial ideas of formulating the *FSS* problems with limited-buffer storage (including blocking, no-wait, limited-buffer constraints) but the proposed algorithm was ineffective. Ronconi (2004) introduced two two-stage hybrid heuristic algorithms to solve the *FSS* with blocking constraints based on the framework of the well-known *NEH* algorithm (Nawaz, Ensco, and Ham 1983). Grabowski and Pempera (2007) developed a tabu search metaheuristic algorithm to solve the *FSS* with blocking constraints. Fink and Voß (2003) developed several metaheuristic algorithms for the *FSS* with no-wait constraints and evaluated the trade-off between running time and solution quality for calibrating the algorithms. Qian et al. (2009) designed a hybrid differential evolution algorithm to solve the *FSS* with limited-buffer constraints. Fu, Sivakumar, and Li (2012) developed a hybrid differential evolution algorithm to solve the *FSS* with intermediate buffers and batch processors. Davendra and Bialic-Davendra (2013) proposed a discrete self-organising migrating algorithm for solving the *FSS* with blocking in an efficient way. Ding et al. (2015) analysed the block properties of the *FSS* with blocking and developed an iterated greedy algorithm to solve the problem efficiently. Zhang et al. (2017) combined a greedy heuristic and a hybrid differential evolution algorithm to solve the *FSS* with a batch processor and limited buffers. Han et al. (2016) proposed a so-called modified fruit fly optimisation algorithm to solve the *FSS* with blocking and demonstrated its efficiency based on benchmark instances.

Compared to the literature of the classical *JSS* problem, the *JSS* problem with no-wait and no-buffer (blocking) constraints received much less attentions. This paper focuses on the *JSS* with various types of buffering constraints. For ease of presentation, the *JSS* problems with a single buffering constraint (i.e. no-wait, blocking, limited-buffer) are, respectively, called *NWJSS*, *BJSS* and *LBJSS* throughout this paper. An initial introduction of *NWJSS* and *BJSS* was given by (Hall and Sriskandarajah 1996). Song and Lee (1998) developed a Petri-net time-marked graph to analyse the deadlock detection properties of *BJSS*. Mati, Rezg, and Xie (2001) investigated an automated manufacturing system, in which the deadlock-prone characteristics in a job shop were analysed based on an extended disjunctive graph. Mati, Lahlou, and Dauzère-Pérès (2011) extended the *BJSS* problem by incorporating more features in a more flexible manufacturing system. Mascis and Pacciarelli (2002) studied the *BJSS* and *NWJSS* problems by means of an alternative graph, which is an extension of classical disjunctive graph. Pacciarelli (2002) further applied the alternative graph to model a complex factory scheduling problem that incorporates the no-wait constraint and sequence-independent set-up times. Schuster and Framinan (2003) developed a hybrid genetic algorithm and simulated annealing metaheuristic algorithm for solving *NWJSS*. Hauptman and Jovan (2004) investigated a real-world process manufacturing system by transforming it into a job shop with no-wait and no-buffer constraints. Brucker and Kampmeyer (2008) proposed a tabu search metaheuristic for a cyclic *BJSS* problem by developing a recovering procedure in neighbourhood moves. Chien, Tseng, and Chen (2008) modelled a patient scheduling problem as *NWJSS* and solved it by an evolutionary algorithm. Gröflin and Klinkert (2009) developed a tabu search algorithm to solve *BJSS* with a special mechanism of satisfying the blocking constraints based on an extended disjunctive graph. Gröflin, Pham, and Bürgy (2011) presented a local search heuristic to solve the flexible *BJSS* problem with the transfer and set-up times, based on the earlier work of (Gröflin and Klinkert 2006, 2009) regarding the construction of a feasible neighbourhood structure. Santosa, Budiman, and Wiratno (2011) developed a hybrid metaheuristic called CEGA (cross-entropy with genetic algorithm) to solve *NWJSS*. Samarghandi and ElMekkawy (2013) developed a genetic algorithm to solve *NWJSS* with sequence-dependent set-up times and single-server constraints. Pranzo and Pacciarelli (2016) developed an iterated greedy algorithm to solve two variants of *BJSS* without/with swap allowed. Brucker et al. (2006) investigated three types of *LBJSS* by classifying buffers into three categories: (i) machine-dependent output buffers; (ii) machine-dependent input buffers; (iii) job-dependent buffers. Witt and Voß (2007) developed three heuristic algorithms to guarantee finding the high-quality *LBJSS* schedule and indicated that the proposed approaches are functional to control the work-in-process operations that wait for processing in the production system with limited intermediate storage. Zeng, Tang, and Yan (2014) extended the *BJSS* problem

using a limited number of automated guided vehicles (AGV) to transfer jobs between machines. The so-called BJS-AGV problem was solved by a two-stage heuristic algorithm based on the analysis of characteristics of this problem. Louaqad and Kamach (2016) investigated the blocking and no-wait job shop scheduling problems in robotic cells and developed a MILP model for solving this complex problem up to 10 jobs, 10 machines and 3 robots.

Based on the above literature review, it is noted that a combination of four buffering constraints in job shop environments was rarely found. To fill this gap, a new scheduling problem called *CBJSS* (Job Shop Scheduling with a Combination of four Buffering constraints) is investigated in this study, which contributes to adding new knowledge in scheduling theory. With a thorough analysis of problem properties, a novel heuristic algorithm that consists of several interactive sub-algorithms, is developed to solve this complicated problem in a very efficient way. The proposed CBJSS model can be used as a fundamental tool to identify, analyse, configure and evaluate different types of scheduling systems that should consider various buffering requirements simultaneously. The proposed solution approach is useful for many real-world applications in food, chemical, automation, railway, aviation and health care industries because diverse buffering conditions occur frequently in these scheduling systems.

The remainder of this paper is organised as follows. In Section 2, the *CBJSS* problem is defined and its problem properties are analysed. In Section 3, a mixed integer programming model of CBJSS is formulated. An efficient heuristic algorithm is developed in Section 4. Extensive computational experiments are reported in Section 5. Contribution and significance of this study are concluded in the last section.

2. Definition and analysis

In a *CBJSS* system, there are n independent and non-preemptive jobs that have to be processed on m machines. The objective is to minimise the makespan. Each job consists of at most m operations, each of which must be processed in a given processing route, but this route may differ with jobs. Only one operation can be processed on one machine and each machine can exactly process one operation at a time. Each machine should be associated with a specified buffering constraint, due to technical, safety or service requirements. In comparison to the processing, blocking times or storing times, the transferring times of jobs between machines (storage units) are negligible and thus omitted in this study. For convenience, parameter $b_i \in \{\varepsilon, \emptyset, \theta|\tau_i, \infty\}$ is defined to represent an intermediate buffering constraint associated with Machine i . If $b_i = \varepsilon$, this buffering constraint is defined as ‘no-wait’, which implies that any job completed on Machine i should be continuously processed without any delay. If $b_i = \emptyset$, this buffering constraint is defined as ‘no-buffer’, which means that there is no buffer storage to store any job after completed on Machine i . If $b_i = \theta|\tau_i$, this buffering constraint is defined as ‘limited-buffer’ if $0 < \tau_i < n$. If $b_i = \infty$, this buffering constraint is defined as ‘infinite-buffer’ if $\tau_i \geq n$. Note that the limited-capacity buffers in our proposed *CBJSS* problem are ‘machine-dependent output buffering storage’, which means that a storage unit may store a job just after its processing on the associated machine. In a sense, these buffer storage units could be treated as ‘dummy parallel machines’ but one main difference is that the utilisation of each buffer storage unit depends on dynamic scheduling scenarios. For more discussions between machines and output buffers, please refer to a book chapter by Brucker and Knust (2012). By extending three-tuple descriptor in scheduling theory, the *CBJSS* problem is denoted as $J_m(b_i \in \{\varepsilon, \emptyset, \theta|\tau_i, \infty\}, i = 1, 2, \dots, m)|n|C_{\max}$, where J_m represents a job shop with m machines; n is the number of jobs; C_{\max} is the makespan; $b_i \in \{\varepsilon, \emptyset, \theta|\tau_i, \infty\}, i = 1, 2, \dots, m$ defines a flexible combination of four different buffering constraints associated with each machine. If the descriptor is $Jm|n|C_{\max}|b_i = \emptyset, \forall i = 1, 2, \dots, m$, then this problem (system) is regarded as the *BJSS* problem (system), which is an extreme case of the *CBJSS* as all of buffering requirements associated with all machines are defined as ‘no-buffer’. If the descriptor is $Jm|n|C_{\max}|b_i = \varepsilon, \forall i = 1, 2, \dots, m$, then this problem (system) is treated as the *NWJSS* problem (system), which is also extreme case of the *CBJSS* as all of buffering requirements associated with all machines are defined as ‘no-wait’.

To analyse, formulate and solve the *CBJSS*, the following notations are defined.

Indices and parameters

n	number of jobs
m	number of machines
j	job index, $j = 1, 2, \dots, n$
J	a set of jobs
J_j	Job j ; $J_j \in J$
i	machine index, $i = 1, 2, \dots, m$
M	a set of machines

M_i	machine i ; $M_i \in M$
π_j	number of operations for job j
o	operation's order index ($o = 1, 2, \dots, \pi_j$) of job j in the given processing route
O_{oj}	the o th operations of job j
h_{oji}	1, if O_{oj} requires M_i in the given processing route; 0, otherwise
P_{oj}	processing time of O_{oj}
b_i	a buffering constraint associated with Machine i
α_i	1, if the buffering constraint of M_i is <i>no-wait</i> ($b_i = \varepsilon$); or 0, otherwise
β_i	1, if the buffering constraint of M_i is <i>no-buffer</i> ($b_i = \phi$); or 0, otherwise
γ_i	1, if the buffering constraint of M_i is <i>limited-buffer</i> ($b_i = \theta$); or 0, otherwise
δ_i	1, if the buffering constraint of M_i is <i>infinite-buffer</i> ($b_i = \infty$); or 0, otherwise
τ_i	number of buffers associated with Machine i when $b_i = \theta$
k	buffer index, $k = 1, 2, \dots, \tau_i$
H	a large constant positive value

Variables

$y_{oj'oj''}$	1, if O_{oj} proceeds $O_{oj''}$; or 0, otherwise
$w_{oj'oj''k}$	1, if O_{oj} proceeds $O_{oj''}$ in buffer $k \in \{1, 2, \dots, \tau_i\}$ of M_i ; or 0, otherwise
z_{ojk}	1, if O_{oj} is stored in buffer $k \in \{1, 2, \dots, \tau_i\}$ after its completion on M_i ; or 0, otherwise
E_{oj}	starting time of O_{oj}
C_{oj}	completion time of O_{oj} ; $C_{oj} = E_{oj} + P_{oj}$
B_{oj}	blocking time of O_{oj}
D_{oj}	departure time of O_{oj} ; $D_{oj} = C_{oj} + B_{oj}$
S_{ojk}	storing time of O_{oj} in buffer $k k = 1, 2, \dots, \tau_i$ of M_i
L_{ojk}	leaving time of O_{oj} in buffer $k k = 1, 2, \dots, \tau_i$; $L_{ojk} = D_{oj} + S_{ojk}$
C_{\max}	maximum completion time or makespan
A_k	the current available time of buffer $k k \in \{1, 2, \dots, \tau_i\}$ of M_i when $b_i = \theta$
A_i^*	the earliest buffer available time of M_i ; $A_i^* = \min_{k \in \{1, 2, \dots, \tau_i\}} A_k$ when $b_i = \theta$
k^*	the assigned buffer with the earliest buffer available time; $k^* = \operatorname{argmin}_{k \in \{1, 2, \dots, \tau_i\}} A_k$ when $b_i = \theta$

Figure 1 is drawn to elucidate the time elements of an operation O_{oj} , including starting time, processing time, completion time, blocking time, departure time, storing time and leaving time. In Figure 1, the values of horizontal axis are measured in timing units (e.g. minutes); the labels of vertical axis are described by each machine together with its buffering constraint indicated by $b_i \in \{\varepsilon, \emptyset, \theta, \tau_i, \infty\}$.

In the following, critical properties of the CBJSS problem are thoroughly analysed.

Property 1: If $b_i = \emptyset$, then $B_{oj} = \max(0, E_{o+1,j} - C_{oj})$.

Analysis: Illustrated in Figure 2, blocking time B_{oj} of operation O_{oj} equals the gap between its completion time C_{oj} and its same-job successor's starting time $E_{o+1,j}$, if the buffering constraint associated with M_i is *no-buffer*, i.e. $b_i = \emptyset$. Due to the absence of buffer storage, M_i has to be blocked until the job is able to be transferred when the downstream machine becomes available.

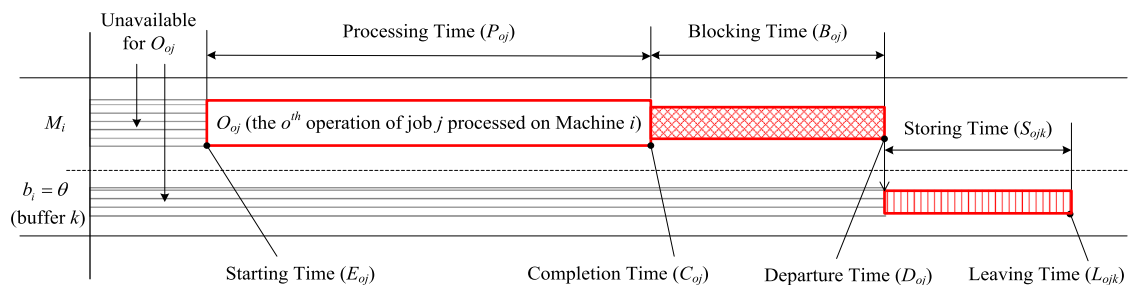


Figure 1. Time elements of an operation in CBJSS.

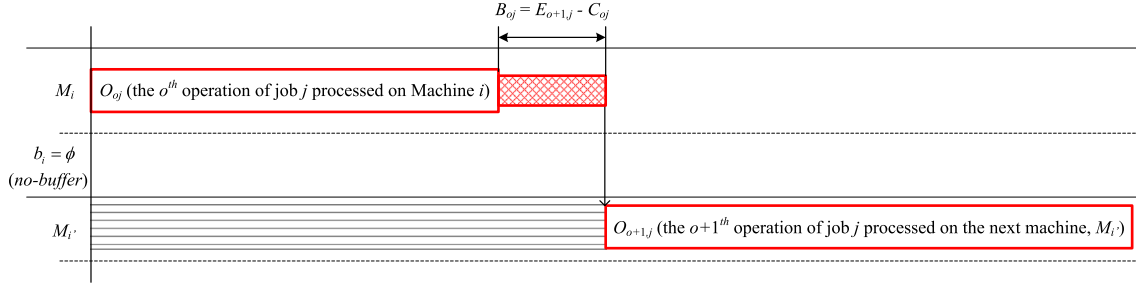


Figure 2. Illustration of Property 1.

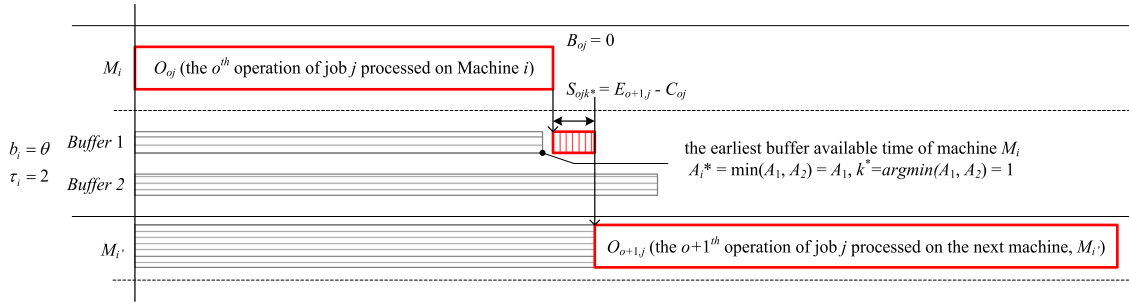


Figure 3. Illustration of Property 3.

Property 2: If $b_i = \theta$ and $C_{oj} = E_{o+1,j}$, then $B_{oj} = 0$ and $S_{ojk^*} = 0$.

Analysis: If the buffering constraint associated with M_i is *limited-buffer* (i.e. $b_i = \theta$) and the completion time C_{oj} of operation O_{oj} is equal to its same-job successor's starting time $E_{o+1,j}$, then the blocking time B_{oj} is zero. In this case, its storing time is also zero as none of buffers of M_i are required.

Property 3: If $b_i = \theta$ and $C_{oj} < E_{o+1,j}$ and $C_{oj} \geq A_i^*$, then $B_{oj} = 0$ and $S_{ojk^*} = E_{o+1,j} - C_{oj}$.

Analysis: As shown in Figure 3, if the buffering constraint associated with M_i is *limited-buffer* (i.e. $b_i = \theta$) and its completion time C_{oj} is less than its **same-job successor's** starting time $E_{o+1,j}$ but greater than or equal to the earliest buffer time A_i^* of M_i , then blocking time B_{oj} of operation O_{oj} is zero because this job can be immediately transferred to an available buffer (i.e. buffer k^*) after its completion on M_i . Moreover, the storing time S_{ojk^*} equals the gap between its same-job successor's starting time $E_{o+1,j}$ and its completion time C_{oj} .

Property 4: If $b_i = \theta$ and $C_{oj} < E_{o+1,j}$ and $C_{oj} < A_i^*$ and $A_i^* < E_{o+1,j}$, then $B_{oj} = A_i^* - C_{oj}$ and $S_{ojk^*} = E_{o+1,j} - A_i^*$.

Analysis: As illustrated in Figure 4, if the buffering constraint associated with M_i is *limited-buffer* (i.e. $b_i = \theta$) and its completion time C_{oj} is less than both its successor's starting time $E_{o+1,j}$ and the earliest buffer time A_i^* , in addition, the earliest buffer time A_i^* is less than its successor's starting time $E_{o+1,j}$, then B_{oj} equals the gap between its completion time C_{oj} and the earliest buffer time A_i^* , implying that M_i has to be blocked till the earliest availability of buffer k^* . After the blocking duration on M_i , this job can be transferred to buffer k^* with the earliest buffer time A_i^* . Thus, the storing time S_{ojk^*} of O_{oj} is the gap between A_i^* and $E_{o+1,j}$. The buffer k^* becomes available when the next machine in the processing route of this job becomes available.

Property 5: If $b_i = \theta$ and $C_{oj} < E_{o+1,j}$ and $C_{oj} < A_i^*$ and $A_i^* \geq E_{o+1,j}$, then $B_{oj} = E_{o+1,j} - C_{oj}$ and $S_{ojk^*} = 0$.

Analysis: As illustrated in Figure 5, if the buffering constraint associated with M_i is *limited-buffer* (i.e. $b_i = \theta$) the earliest buffer time A_i^* is greater than or equal to its completion time C_{oj} and its successor's starting time $E_{o+1,j}$, none the buffers of M_i are available to be used. In this scenario, the storing time S_{ojk^*} of O_{oj} must be zero and its blocking time B_{oj} equals the gap between C_{oj} and $E_{o+1,j}$.

The following properties are analysed for the scenarios when the buffering constraint associated with M_i is *no-wait* (i.e. $b_i = \varepsilon$).

Property 6: If $b_i = \varepsilon$, then $B_{oj} = 0$ and $C_{oj} = E_{o+1,j}$.

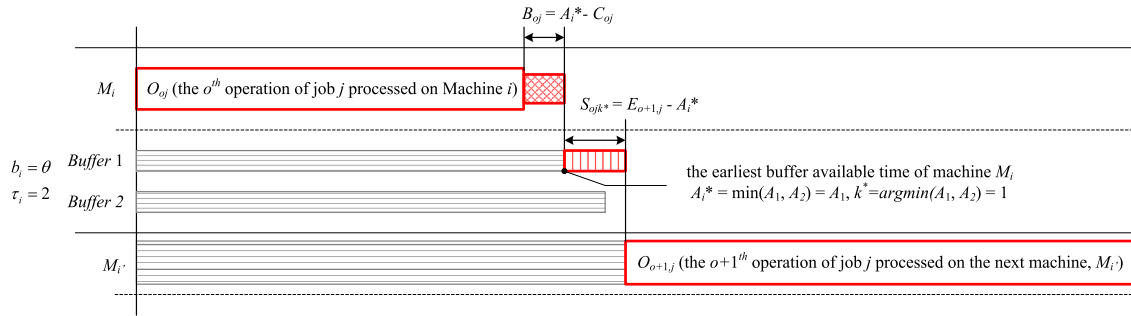


Figure 4. Illustration of Property 4.

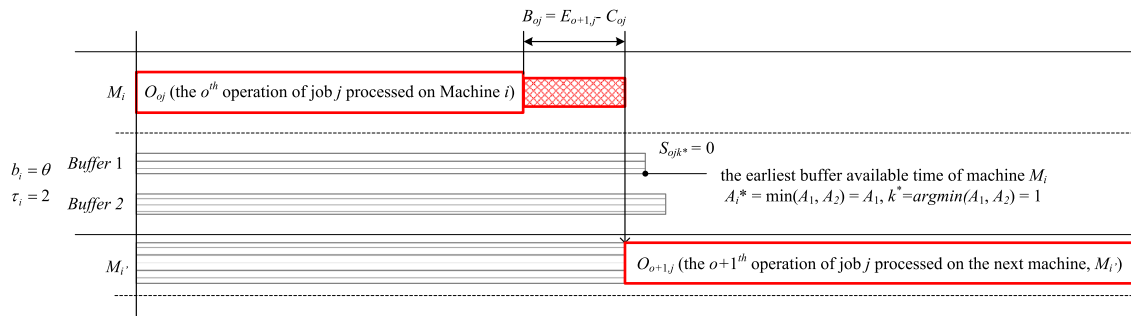


Figure 5. Illustration of Property 5.

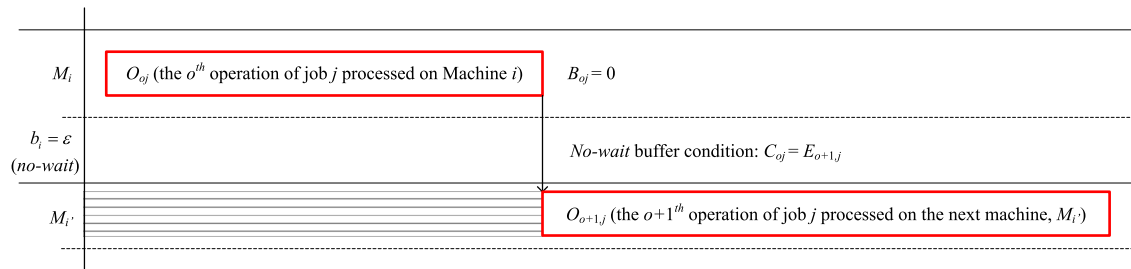


Figure 6. Illustration of Property 6.

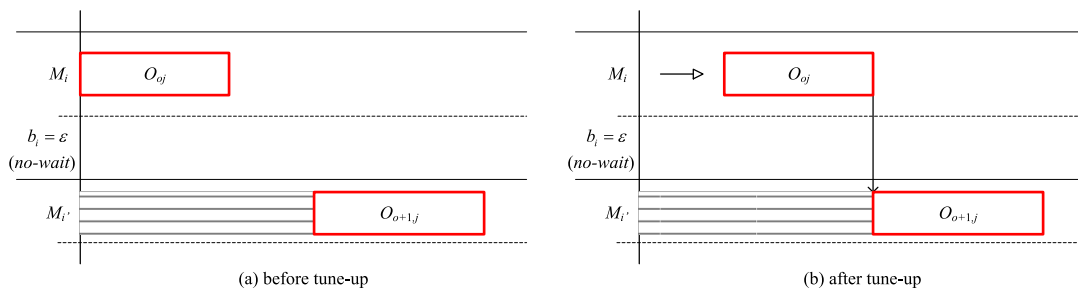


Figure 7. Illustration of Property 7.

Analysis: As shown in Figure 6, the blocking time of O_{oj} must be zero, if the buffering constraint associated with M_i is *no-wait*. This implies that this job must be immediately processed on the next machine without any delay due to the requirement of *no-wait* constraint.

Property 7: In case $b_i = \varepsilon$ and $C_{oj} < E_{o+1,j}$, the tune-up procedure should be applied to update $E_{oj} = E_{o+1,j} - P_{oj}$ to satisfy the *no-wait* constraint.

Analysis: As illustrated in Figure 7, the tune-up procedure (Liu and Kozan 2009b) may be applied to satisfy the *no-wait* constraint in an iterated way. In this case, the starting time E_{oj} of O_{oj} is re-calculated and equals the gap between its same-job successor's starting time $E_{o+1,j}$ and its processing time P_{oj} .

Property 8: If $(E_{o'j'} > E_{oj}$ and $E_{o'j'} < D_{oj})$ or $(E_{oj} > E_{o'j'}$ and $E_{oj} < D_{o'j'})$, two conflicting scenarios may incur in the solution procedure.

Analysis: As shown in Figure 8(a1), one conflicting scenario happens when the starting time of $O_{o'j'}$ is greater than the starting time of O_{oj} but less than the departure time of O_{oj} . On the other hand, as shown in Figure 8(b1), the starting time of O_{oj} is greater than the starting time of $O_{o'j'}$ but less than the departure time of $O_{o'j'}$.

The following additional notations are particularly defined for Property 9 to be used only in the proposed heuristic algorithm in Section 4 (Figure 9).

Additional Parameters for Property 9

- R_{oj} ready time of operation O_{oj} of job J_j that is currently considered in the solution procedure
- n_i number of operations that have already been scheduled on M_i at the current stage
- $[x]$ sequence index of the x th $x = 1, \dots, n_i$ operation already scheduled on M_i
- $O_{[x],i}$ the x th operation already scheduled on M_i
- $E_{[x],i}$ starting time of $O_{[x],i}$ already scheduled on M_i
- $P_{[x],i}$ processing time of $O_{[x],i}$ already scheduled on M_i
- $D_{[x],i}$ departure time of $O_{[x],i}$ already scheduled on M_i

Additional Variables for Property 9

- $E_{oj}^{[\lambda]}$ the earliest starting time of O_{oj} with the best insertion position λ
- λ the best insertion position index to insert a new operation O_{oj} into the current sequence of operations already scheduled on M_i

Property 9: If $E_{[1],i} - R_{oj} \geq P_{oj}$, then $E_{oj}^{[1]} = R_{oj}$ and $\lambda = 1$; else if $E_{[1],i} - R_{oj} < P_{oj}$ and $\min_{x \in \{2, \dots, n_i\}} (E_{[x],i} - R_{oj}, E_{[x],i} - D_{[x-1],i}) \geq P_{oj}$, then $\lambda = \operatorname{argmin}_{x \in \{2, \dots, n_i\}} (\max(R_{oj}, D_{[x-1],i}))$ and $E_{oj}^{[\lambda]} = \max(R_{oj}, D_{[\lambda-1],i})$; else if $E_{[1],i} - R_{oj} < P_{oj}$ and $\min_{x \in \{2, \dots, n_i\}} (E_{[x],i} - R_{oj}, E_{[x],i} - D_{[x-1],i}) < P_{oj}$, then $\lambda = n_i + 1$ and $E_{oj}^{[\lambda]} = \max(R_{oj}, D_{[n_i],i})$.

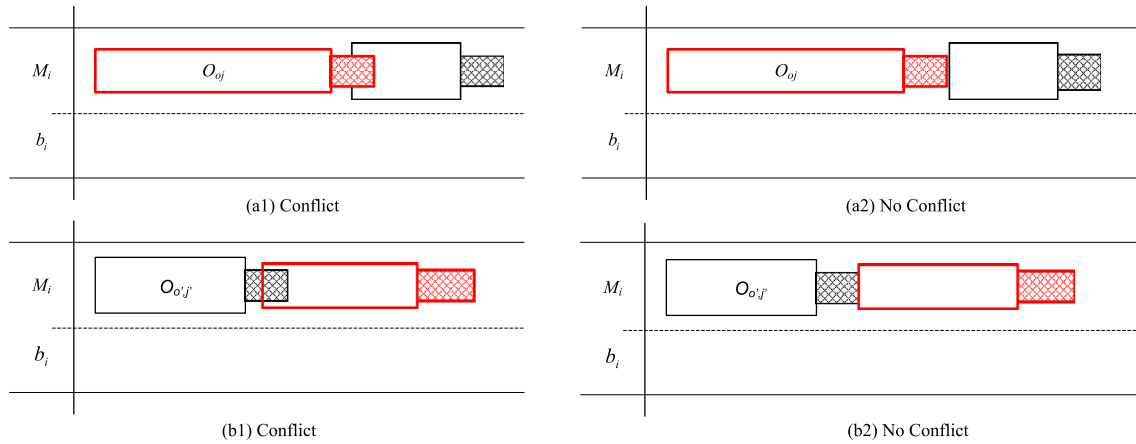


Figure 8. Illustration of Property 8.

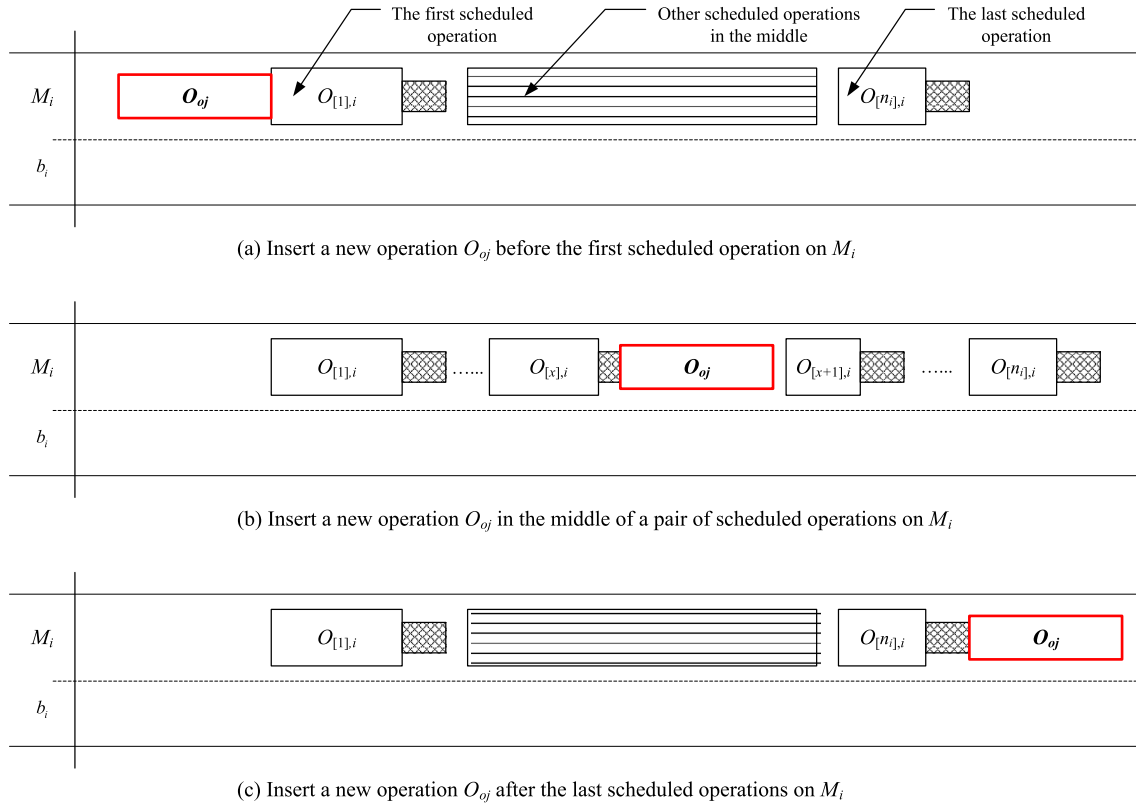


Figure 9. Illustration of Property 9.

3. Mathematical formulation

Based on the analysis of problem properties, the CBJSS is mathematically formulated below.

3.1 CBJSS formulation

Minimise:

$$C_{\max} \quad (1)$$

The objective function is to minimise the maximum completion time, i.e. makespan.

Subject to:

$$H(2 - h_{oji} - h_{o'j'i}) + My_{oj'o'j'} + E_{oj} \geq E_{o'j'} + P_{o'j'} + B_{o'j'} \quad (2)$$

$$H(2 - h_{oji} - h_{o'j'i}) + M(1 - y_{oj'o'j'}) + E_{o'j'} \geq E_{oj} + P_{oj} + B_{oj} \quad (3)$$

$$h_{oji} + h_{o'j'i} - 1 \geq y_{oj'o'j'} \quad (4)$$

$$o = 1, 2, \dots, \pi_j - 1; \quad o' = 1, 2, \dots, \pi_{j'} - 1; \quad j, j' = 1, 2, \dots, n | j \neq j'; \quad i = 1, 2, \dots, m$$

Constraints (2–4) satisfy the exclusive precedence relationship of a pair of operations (i.e. O_{oj} and $O_{o'j'}$) that belong to two jobs, respectively, (i.e. J_j and $J_{j'}$).

$$\sum_{i=1}^m h_{oji} (E_{oj} + P_{oj}) \leq \sum_{i=1}^m h_{o+1,j,i} E_{o+1,j} \quad (5)$$

$$o = 1, 2, \dots, \pi_j - 1; \quad j = 1, 2, \dots, n;$$

Constraints (5) requires that the completion time of an operation O_{oj} must be less than or equal to the starting time of its same-job successor $O_{o+1,j}$, whatever the buffering constraint is.

$$\sum_{i=1}^m \alpha_i h_{oji} (E_{oj} + P_{oj}) = \sum_{i=1}^m h_{o+1,j,i} E_{o+1,j} \quad (6)$$

$$\sum_{i=1}^m \alpha_i x_{oji} B_{oj} = 0 \quad (7)$$

$$o = 1, 2, \dots, \pi_j - 1; \quad j = 1, 2, \dots, n;$$

Constraints (6–7) define the *no-wait* constraint of M_i excluding the last machine in the processing route of each job. In this case, the completion time of O_{oj} must be equal to the starting time of its same-job successor $O_{o+1,j}$. Constraint (7) requires that the blocking time of O_{oj} must be zero under the *no-wait* requirement.

$$\sum_{i=1}^m \beta_i h_{oji} (E_{oj} + P_{oj} + B_{oj}) = \sum_{i=1}^m h_{o+1,j,i} E_{o+1,j}$$

$$o = 1, 2, \dots, \pi_j - 1; \quad j = 1, 2, \dots, n \quad (8)$$

Constraint (8) defines the *no-buffer* constraint of M_i excluding the last machine. In this case, after the completion of O_{oj} , M_i may be blocked due to the absence of associated buffers. Thus, the blocking time (B_{oj}) of O_{oj} may be non-zero.

$$\sum_{i=1}^m \gamma_i h_{oji} (E_{oj} + P_{oj} + B_{oj} + S_{ojk}) = \sum_{i=1}^m h_{o+1,j,i} E_{o+1,j} \quad (9)$$

$$D_{oj} = E_{oj} + P_{oj} + B_{oj} \quad (10)$$

$$\sum_{i=1}^m \gamma_i h_{oji} z_{ojk} = 1 \quad (11)$$

$$z_{ojk} + z_{o'j'k} - 1 \geq w_{oj'o'j'k} \quad (12)$$

$$H(2 - z_{ojk} - z_{o'j'k}) + M w_{oj'o'j'k} + D_{oj} \geq D_{o'j'} + S_{o'j'k} \quad (13)$$

$$H(2 - z_{ojk} - z_{o'j'k}) + M(1 - w_{oj'o'j'k}) + D_{o'j'} \geq D_{oj} + S_{ojk} \quad (14)$$

$$o = 1, 2, \dots, \pi_j - 1; \quad o' = 1, 2, \dots, \pi_{j'} - 1; \quad j, j' = 1, \dots, n | j \neq j'; \quad k = 1, 2, \dots, \tau_i;$$

Constraints (9–14) define the *limited-buffer* constraint of M_i excluding the last machine. In this case, after the completion of O_{oj} , its leaving time in buffer k must be equal to the starting time of its same-job successor $O_{o+1,j}$ as its storing time may be non-zero. Moreover, only one buffer k ($k \in 1, 2, \dots, \tau_i$) of M_i can be used to store O_{oj} at a time. If both operations O_{oj} and $O_{o'j'}$ require the same buffer k , i.e. $z_{ojk} = z_{o'j'k} = 1$, then the precedence relationship between them is defined. In a sense, Constraints (13–14) imply that the limited-capacity intermediate buffers (storing units) could be regarded as the *dummy* machines that may not be used. If the storing time of O_{oj} is non-zero, it means that a storage unit is used by O_{oj} .

$$\sum_{i=1}^m \delta_i h_{\pi_j,j,i} (E_{\pi_j,j} + P_{\pi_j,j}) \leq C_{\max} \quad (15)$$

$$\sum_{i=1}^m \delta_i h_{\pi_j,j,i} B_{\pi_j,j} = 0 \quad (16)$$

$$j = 1, 2, \dots, n$$

Constraints (15–16) define the *infinite-buffer* constraint of the last machine in the processing route of each job. In this case, blocking time of the last operation must be zero, as the totally finished job (the final product) is allowed to be immediately removed from the production environment. In addition, it needs to satisfy maximum completion time constraints, i.e. the completion time of the last (π_j th) operation $O_{\pi_j,j}$ of J_j should not exceed the makespan.

$$E_{oj}, B_{oj}, C_{oj}, D_{oj}, S_{ojk} \geq 0 \quad (17)$$

$$y_{oj o' j'}, z_{ojk}, w_{ojk o' j' k} \in \{0, 1\} \quad (18)$$

$$o = 1, 2, \dots, \pi_j; o' = 1, 2, \dots, \pi_{j'}; j, j' = 1, \dots, n | j \neq j'; k = 1, 2, \dots, \tau_i;$$

Constraints (17–18) declare non-negativity and binary constraints, respectively.

4. Solution approach

Solving *CBJSS* is a challenge due to the following difficulties: (i) the potential conflicts may be arisen due to the *no-buffer* constraint, especially when two operations on a blocked machine are not allowed to be swapped; (ii) the *no-wait* constraint is so restrictive that the starting times of the same-job predecessors of an operation may need to be reversely tuned up in an iterated way; (iii) the *limited-buffer* constraint requires the determination of the earliest available buffer in a dynamic way. Based on Properties 1–9, an innovative heuristic algorithm (called the *CBJSS-CA-BIH* algorithm) is developed to obtain a high-quality *CBJSS* solution in an efficient way. In the framework of best-insertion-heuristic (*BIH*), a complicated constructive algorithm (called the *CBJSS-CA* algorithm) is developed and embedded inside the *CBJSS-CA-BIH* algorithm.

A small-size example is given to illustrate the *CBJSS-CA-BIH* heuristic algorithm. In this example, it is assumed that the current partial sequence of scheduled jobs is $\Pi = \{J_1 \rightarrow J_2 \rightarrow J_3\}$; the list of unscheduled jobs is $U = \{J_4, J_5\}$. By

CBJSS-CA algorithm

Step 1: For each job J_j in sequence, get the number of operations for J_j in an alternative: π_j .

Step 2: Initialise the order index of the current operation of J_j : $o \leftarrow 1$.

Step 3: While ($o \leq \pi_j$)

3.1 Get the machine that is assigned to process operation O_{oj} : M_i .

3.2 Determine the earliest starting time of O_{oj} with the best insertion position λ , by inserting O_{oj} into the current sequence of operations that have been already scheduled on M_i based on Property 9: $E_{oj}^{[\lambda]}$ and λ .

3.3 If $o > 1$, determine the blocking time $B_{o-1,j}$ and the storing time $S_{o-1,j}$ of $O_{o-1,j}$ (i.e. the same-job predecessor of O_{oj}) in terms of the given buffering constraint (i.e. defined by the value of b_{i-1}) of M_{i-1} :

3.3.1 If $b_{i-1} = \emptyset$ (i.e. ‘no-buffer’), then apply Property 1 to set $B_{o-1,j} \leftarrow \max(0, E_{oj} - C_{o-1,j})$; $S_{o-1,j} \leftarrow 0$.

3.3.2 Else if $b_{i-1} = \theta$ (i.e. ‘limited-buffer’), then apply Properties 2–5 in terms of the following four scenarios.

3.3.2.1 If $C_{o-1,j} = E_{oj}$, then set $B_{o-1,j} \leftarrow 0$; $S_{o-1,j} \leftarrow 0$.

3.3.2.2 Else if $C_{o-1,j} < E_{oj}$ and $C_{o-1,j} \geq A_{i-1}^*$, then set $B_{o-1,j} \leftarrow 0$; $S_{o-1,jk^*} \leftarrow E_{oj} - C_{o-1,j}$; $S_{o-1,j} \leftarrow S_{o-1,jk^*}$.

3.3.2.3 Else if $C_{o-1,j} < E_{oj}$ and $C_{o-1,j} < A_{i-1}^*$ and $A_{i-1}^* < E_{oj}$, then set $B_{o-1,j} \leftarrow A_{i-1}^* - C_{o-1,j}$; $S_{o-1,jk^*} \leftarrow E_{oj} - A_{i-1}^*$;

$S_{o-1,j} \leftarrow S_{o-1,jk^*}$.

3.3.2.4 Else if $C_{o-1,j} < E_{oj}$ and $C_{o-1,j} < A_{i-1}^*$ and $A_{i-1}^* \geq E_{oj}$, then set $B_{o-1,j} \leftarrow E_{oj} - C_{o-1,j}$; $S_{o-1,j} \leftarrow 0$.

3.3.3 Else if $b_{i-1} = \varepsilon$ (i.e. ‘no-wait’), then apply Properties 6–7 to set

$E_{o-1,j} \leftarrow \max(E_{o-1,j}, E_{oj} - P_{o-1,j})$; $C_{o-1,j} \leftarrow E_{oj}$; $B_{o-1,j} \leftarrow 0$; $S_{o-1,j} \leftarrow 0$.

3.4 Set $D_{o-1,j} \leftarrow C_{o-1,j} + B_{o-1,j}$; $L_{o-1,j} \leftarrow D_{o-1,j} + S_{o-1,j}$.

3.5 Justify the conflicting status based on Property 8.

3.6 If non-conflicting, then set $o \leftarrow o + 1$ and go to Step 3.

3.7 Else, apply the following to eliminate the conflict.

3.7.1 Get the number of scheduled operations on M_{i-1} : n_{i-1} .

3.7.2 For x from 1 to n_{i-1} ,

If $(E_{[x],i-1} \geq E_{o-1,j}$ and $E_{[x],i-1} < D_{o-1,j})$ or $(E_{o-1,j} \geq E_{[x],i-1}$ and $E_{o-1,j} < D_{[x],i-1})$, then update the ready time of $O_{o-1,j}$: $R_{o-1,j} \leftarrow \max(E_{o-1,j}, D_{[x],i-1})$ and break.

3.7.3 Reset the starting time $E_{o-1,j}$ based on the updated $R_{o-1,j}$.

3.7.4 Apply Steps 3.3–3.6 to re-determine time information of same-job predecessors (i.e. from $o - 2$ to 1) of $O_{o-1,j}$ in a reverse order until a new non-conflicting operation at Step 3.6 is found.

3.8 If $o \leq \pi_j$, go to Step 3; else, go to Step 1 for the next job in sequence.

CBJSS-CA-BIH algorithm

Step 1: Set \mathcal{F} which is the initial list of sorted jobs by sorting the jobs in the non-increasing order of the largest sum of the processing times on machines, each of which the buffering constraint is *no-wait*.

Step 2: Initialise Π , which is the partial sequence of scheduled jobs containing the first sorted job in \mathcal{F} .

Step 3: Initialise $U = \mathcal{F} - \Pi$, which is the list of sorted jobs that are unscheduled.

Step 4: While U is not empty,

4.1 For each job $J_j \in U$ in order:

4.1.1 Construct a set of alternatives $A = \bigcup_{(a=1)}^{n_u \times (n_s+1)} \Pi'_a(J_j)$, each of which is a new sequence obtained by adding an unscheduled job $J_j \in U$ into the current partial sequence Π . The number of alternatives equals: $n_A = n_u \times (n_s + 1)$ that is all possible combinations of n_u unscheduled jobs with $n_s + 1$ insertion positions, where n_u is the number of unscheduled jobs in U and n_s is the number of currently scheduled jobs in Π .

4.1.2 For each alternative in A :

4.1.2.1 Apply the *CBJSS-CA* algorithm to construct the feasible *CBJSS* schedule.

4.1.2.2 According to the constructed schedule, get and save the makespan value for each alternative.

4.2 Among all alternatives in A , determine the best alternative that leads to the minimum makespan and the best inserted job denoted as J_{j^*} .

4.3 Update Π that is the best alternative determined in Step 4.2.

4.4 Remove J_{j^*} from U : $U \leftarrow U - \{J_{j^*}\}$.

inserting each unscheduled job in U into the current partial sequence of scheduled jobs Π , a set of alternatives (i.e. a set of new sequences) A is obtained as: $A = [\{J_4 \rightarrow J_1 \rightarrow J_2 \rightarrow J_3\}, \{J_1 \rightarrow J_4 \rightarrow J_2 \rightarrow J_3\}, \{J_1 \rightarrow J_2 \rightarrow J_4 \rightarrow J_3\}, \{J_1 \rightarrow J_2 \rightarrow J_3 \rightarrow J_4\}, \dots, \{J_1 \rightarrow J_2 \rightarrow J_3 \rightarrow J_5\}]$, as analysed in detail as below. In Step 4.1.1, the number of alternatives in the set A is obtained as 8, that is, $n_A = n_u \times (n_s + 1) = 2 \times (3 + 1) = 8$. For each alternative in A , the proposed *CBJSS-CA* construction algorithm is implemented to construct the feasible *CBJSS* schedule in Step 4.1.2. The best alternative with the minimum makespan is selected in Step 4.2 to update the set Π in Step 4.3 and the set U in Step 4.4 for the next iteration.

In the *CBJSS-CA-BIH* algorithm, the number of total alternatives is computed as:

$$\begin{array}{llll}
 \begin{array}{c} J_4, J_5 \\ \downarrow \\ J_1 \rightarrow J_2 \rightarrow J_3 \\ n = 5, n_s = 3, n_u = n - n_s = 2 \\ |A| = n_u \times (n_s + 1) = 8 \end{array} &
 \begin{array}{c} J_4, J_5 \\ \downarrow \\ J_1 \rightarrow J_2 \rightarrow J_3 \end{array} &
 \begin{array}{c} J_4, J_5 \\ \downarrow \\ J_1 \rightarrow J_2 \rightarrow J_3 \end{array} &
 \begin{array}{c} J_4, J_5 \\ \downarrow \\ J_1 \rightarrow J_2 \rightarrow J_3 \end{array} \\
 J = \{J_1, J_2, J_3, J_4, J_5\} & & & \\
 \Pi'_1(J_4) = \{J_4 \rightarrow J_1 \rightarrow J_2 \rightarrow J_3\} & & & \\
 \Pi'_2(J_4) = \{J_1 \rightarrow J_4 \rightarrow J_2 \rightarrow J_3\} & & & \\
 \Pi'_3(J_4) = \{J_1 \rightarrow J_2 \rightarrow J_4 \rightarrow J_3\} & & & \\
 \Pi'_4(J_4) = \{J_1 \rightarrow J_2 \rightarrow J_3 \rightarrow J_4\} & & & \\
 \Pi = \{J_1 \rightarrow J_2 \rightarrow J_3\} & & U = \{J_4, J_5\} & \\
 \Pi'_5(J_5) = \{J_5 \rightarrow J_1 \rightarrow J_2 \rightarrow J_3\} & & & \\
 \Pi'_6(J_5) = \{J_1 \rightarrow J_5 \rightarrow J_2 \rightarrow J_3\} & & & \\
 \Pi'_7(J_5) = \{J_1 \rightarrow J_2 \rightarrow J_5 \rightarrow J_3\} & & & \\
 \Pi'_8(J_5) = \{J_1 \rightarrow J_2 \rightarrow J_3 \rightarrow J_5\} & & &
 \end{array}$$

$$\begin{aligned}
 f(n) &= \sum_{n_s=1}^{n_s=n-1} (n - n_s)(n_s + 1) = (n - 1) \sum_{n_s=1}^{n_s=n-1} n_s + \sum_{n_s=1}^{n_s=n-1} n - \sum_{n_s=1}^{n_s=n-1} n_s^2 \\
 &= \frac{(n-1)(n-1)n}{2} + n(n-1) - \frac{(n-1)n(2n-1)}{6} = \frac{1}{6}n^3 + \frac{1}{2}n^2 - \frac{2}{3}n.
 \end{aligned}$$

Computational complexity indicates how much effort is needed to apply an algorithm, estimated by the rate growth function (i.e. $f(n)$) for solution space in terms of problem size. According to this analysis, the asymptotic complexity of the proposed *CBJSS-CA-BIH* heuristic algorithm depends mainly on the first term ($\frac{1}{6}n^3$) due to the cubic growth of the first term as the other two terms can be disregarded for the large n . Hence, the computational complexity of the proposed *CBJSS-CA-BIH* heuristic algorithm is $O(n^3)$.

5. Computational results

To evaluate the performance of the *CBJSS-CA-BIH* algorithm, a collection of benchmark *CBJSS* instances are established based on Lawrence's *JSS* data from OR-Library (Beasley 1990). In the literature, the Lawrence's *JSS* instances (40 instances in total) are well known as the difficult benchmarks. To differentiate the instance types between *JSS* and *CBJSS*, the original Lawrence's data are named *JSS-LA* instances. By keeping the same processing times and the

processing routes of jobs in Lawrence's data as well as adding the specified buffering constraints of machines, the 40 established benchmark data for *CBJSS* are called *CBJSS-LA* (i.e. *CBJSS-LA01* to *CBJSS-LA40*) instances, which are defined in the following way.

- $b_1 = \varepsilon$; $b_2 = \emptyset$; $b_3 = \theta|1$; $b_4 = \theta|2$; $b_5 = \theta|3$ for 5-machine instances (i.e. from *CBJSS-LA01* to *CBJSS-LA15*).
- $b_1 = \varepsilon$; $b_2 = \emptyset$; $b_3 = \theta|1$; $b_4 = \theta|2$; $b_5 = \theta|3$; $b_6 = \varepsilon$; $b_7 = \emptyset$; $b_8 = \theta|1$; $b_9 = \theta|2$; $b_{10} = \theta|3$ for 10-machine instances (i.e. from *CBJSS-LA16* to *CBJSS-LA35*).
- $b_1 = \varepsilon$; $b_2 = \emptyset$; $b_3 = \theta|1$; $b_4 = \theta|2$; $b_5 = \theta|3$; $b_6 = \varepsilon$; $b_7 = \emptyset$; $b_8 = \theta|1$; $b_9 = \theta|2$; $b_{10} = \theta|3$; $b_{11} = \varepsilon$; $b_{12} = \emptyset$; $b_{13} = \theta|1$; $b_{14} = \theta|2$; $b_{15} = \theta|3$ for 15-machine instances (i.e. from *CBJSS-LA36* to *CBJSS-LA40*).

For example, the 10-job, 5-machine *CBJSS-LA01* instance is denoted as $J_5(b_1 = \varepsilon, b_2 = \emptyset, b_3 = \theta|3 = 1, b_4 = \theta|\tau_4 = 2, b_5 = \theta|\tau_5 = 3)|10|C_{\max}$ and the buffering constraints of this instance are explained below:

- the buffering constraint of M_1 is: 'no-wait'
- the buffering constraint of M_2 is: 'no-buffer'
- the buffering constraint of M_3 is: 'limited-buffer' with one buffer
- the buffering constraint of M_4 is: 'limited-buffer' with two buffers
- the buffering constraint of M_5 is: 'limited-buffer' with three buffers

In a *CBJSS* system, it is noted that the buffering constraint of the last machine in the processing route of each job can be neglected as 'infinite-buffer', because the completed job (the final product) is allowed to be immediately removed from the production system.

With integration of a MIP solver' dynamic link library (i.e. IBM ILOG-CPLEX 12.4 for academic use), the MIP model in Section 2 and the *CBJSS-CA-BIH* algorithm in Section 3 were coded together in C# language and solved under Microsoft Visual Studio 2012; and tested on a desktop computer with Intel Core i7 vPro CPU at 3.4 GHz and 8-GB RAM. To illustrate the output of *CBJSS*, a Gantt chart is drawn in Figure 10 to display an optimal schedule of the 10-job, 10-machine *CBJSS-LA20* instance. In Figure 10, the horizontal axis represents the time units (e.g. the makespan of the *CBJSS-LA20* instance is 922), while the vertical axis describes each machine index and its associated buffering status (e.g. the buffering constraint of Machine 1 is *no-wait*).

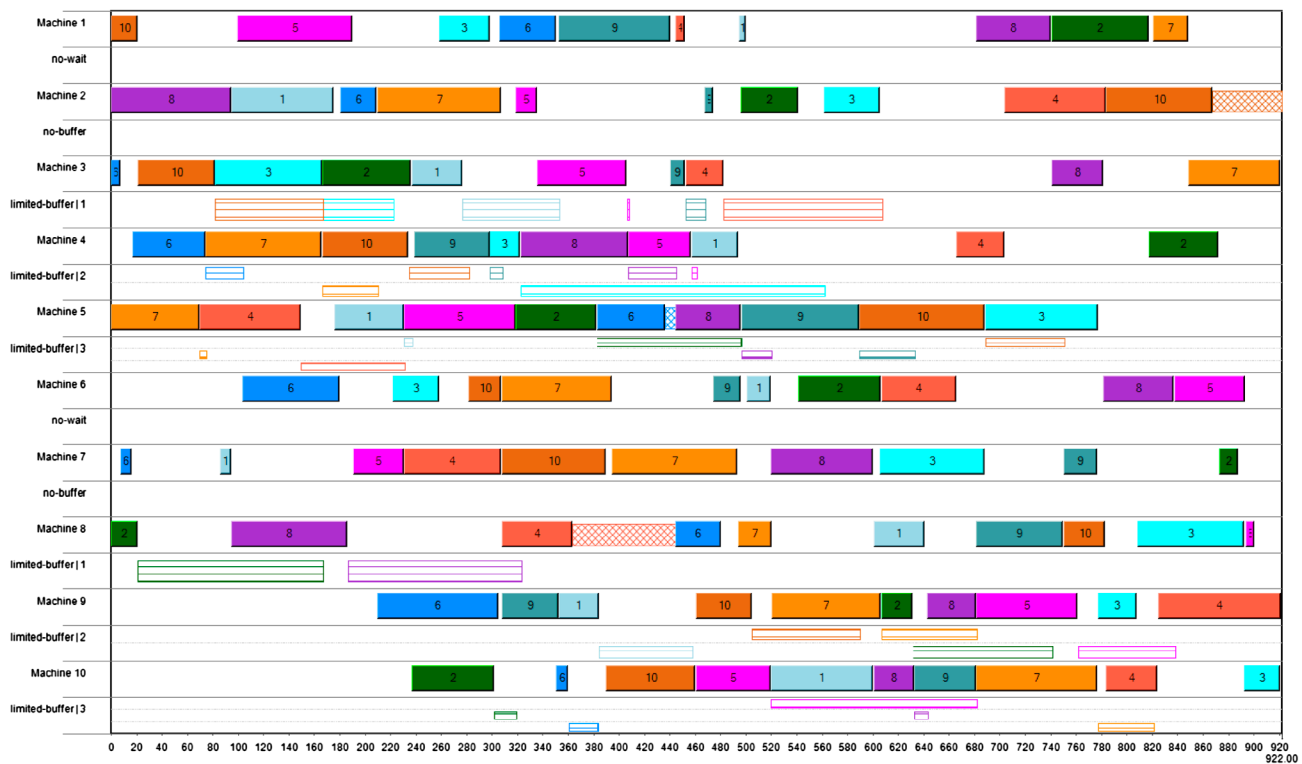
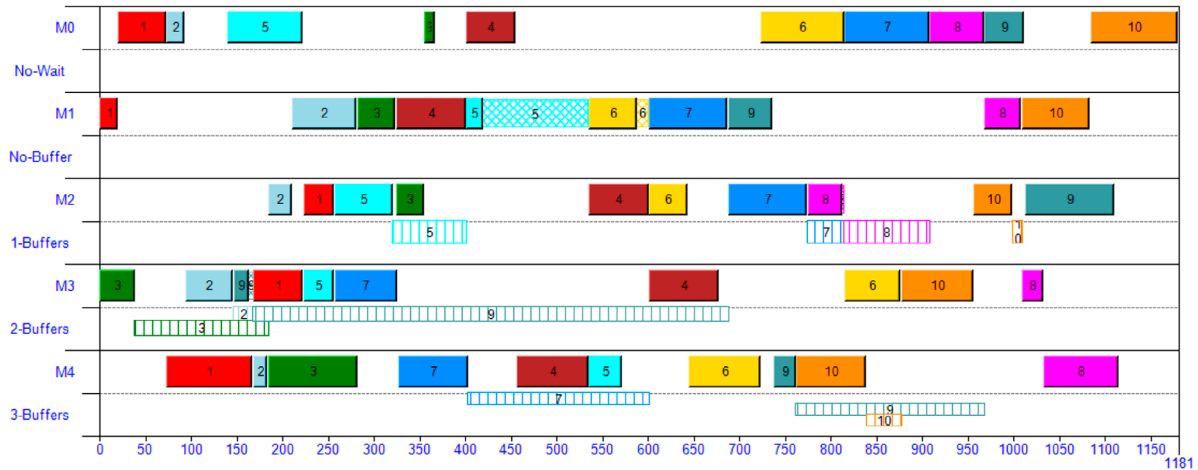
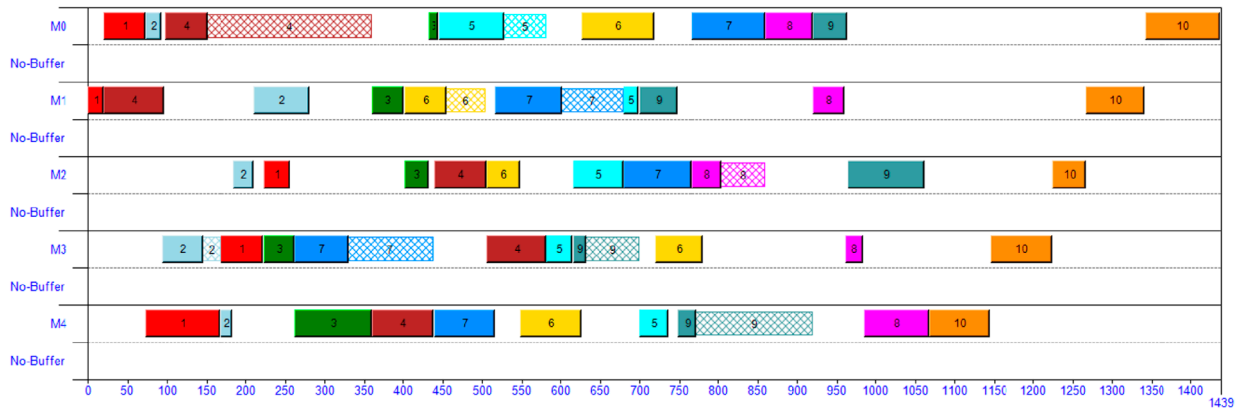


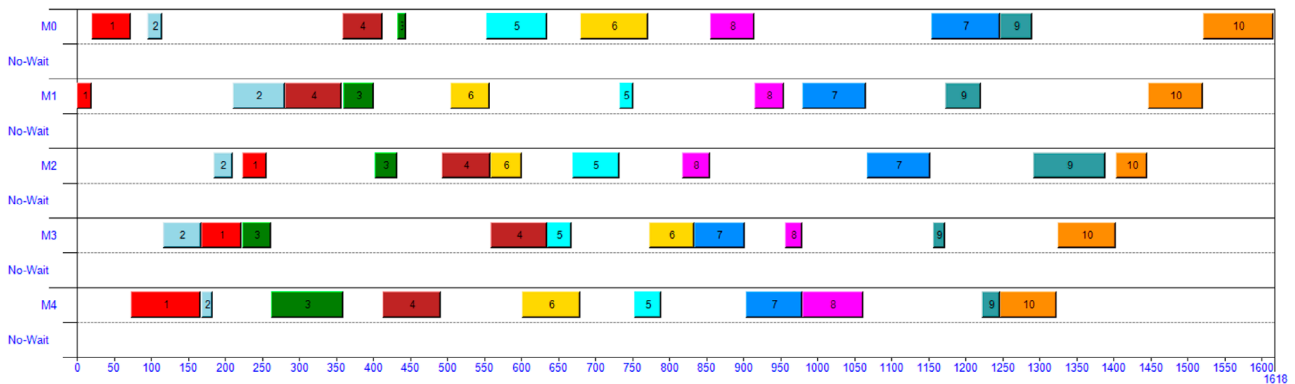
Figure 10. Gantt chart for the optimal schedule of *CBJSS-LA20*.



(a): Gantt chart for the CBJSS_LA01 schedule, of which the makespan is 1181.



(b): Gantt chart for the BJSS_LA01 schedule, of which the makespan is 1439.



(c): Gantt chart for the NWJSS_LA01 schedule, of which the makespan is 1618.

Figure 11. (a) Gantt chart for the CBJSS_LA01 schedule, of which the makespan is 1181. (b) Gantt chart for the BJSS_LA01 schedule, of which the makespan is 1439. (c) Gantt chart for the NWJSS_LA01 schedule, of which the makespan is 1618.

To illustrate the exact application of this CBJSS research, a 10-job, 5-machine sample instance (i.e. CBJSS_LA01) is given below by comparing the CBJSS schedule with the schedules of its corresponding extreme cases (i.e. BJSS_LA01 and NWJSS_LA01), as shown in Figure 11 (a–c). Such a comparative analysis among the CBJSS_LA01,

Table 1. Computational results of Lawrence benchmark CBJSS instances.

Instances	$n*m$	LB^a	ILOG-CPLEX			CBJSS-CA-BIH Algorithm		
			Makespan1	Time1 (s)	Status ^b	Makespan2	Time2 (s)	Gap (%)
CBJSS-LA01	10*5	666	679	14.9	Optimal	718	0.1	5.43
CBJSS-LA02	10*5	635	699	16.2	Optimal	729	0.1	4.12
CBJSS-LA03	10*5	588	670	21.9	Optimal	701	0.1	4.42
CBJSS-LA04	10*5	537	616	61.8	Optimal	657	0.1	6.24
CBJSS-LA05	10*5	593	599	94.2	Optimal	603	0.1	0.66
CBJSS-LA06	15*5	926	978	3600.0	Feasible	1070	0.1	8.60
CBJSS-LA07	15*5	869	989	3600.0	Feasible	1000	0.2	1.10
CBJSS-LA08	15*5	863	995	3600.0	Feasible	1027	0.2	3.12
CBJSS-LA09	15*5	951	1105	3600.0	Feasible	1119	0.2	1.25
CBJSS-LA10	15*5	958	1039	3600.0	Feasible	1104	0.2	5.89
CBJSS-LA11	20*5	1222	1574	3600.0	Feasible	1484	0.3	-6.06
CBJSS-LA12	20*5	1039	1412	3600.0	Feasible	1325	0.4	-6.57
CBJSS-LA13	20*5	1150	1429	3600.0	Feasible	1413	0.4	-1.13
CBJSS-LA14	20*5	1292	1598	3600.0	Feasible	1475	0.4	-8.34
CBJSS-LA15	20*5	1207	1652	3600.0	Feasible	1512	0.4	-9.26
CBJSS-LA16	10*10	717	975	172.2	Optimal	1023	0.3	4.69
CBJSS-LA17	10*10	683	832	150.7	Optimal	920	0.4	9.57
CBJSS-LA18	10*10	663	880	170.6	Optimal	964	0.4	8.71
CBJSS-LA19	10*10	685	878	134.5	Optimal	935	0.4	6.10
CBJSS-LA20	10*10	756	922	138.3	Optimal	1005	0.4	8.26
CBJSS-LA21	15*10	1040	1273	7200.0	Feasible	1294	0.3	1.62
CBJSS-LA22	15*10	830	1120	7200.0	Feasible	1158	0.3	3.28
CBJSS-LA23	15*10	1032	1321	7200.0	Feasible	1453	0.3	9.08
CBJSS-LA24	15*10	857	1333	7200.0	Feasible	1393	0.3	4.31
CBJSS-LA25	15*10	864	1351	7200.0	Feasible	1411	0.3	4.25
CBJSS-LA26	20*10	1218	2602	14400.0	Feasible	1939	0.7	-25.48
CBJSS-LA27	20*10	1235	2808	14400.0	Feasible	1918	0.8	-31.70
CBJSS-LA28	20*10	1216	2518	14400.0	Feasible	1916	0.8	-23.91
CBJSS-LA29	20*10	1120	2537	14400.0	Feasible	1817	0.8	-28.38
CBJSS-LA30	20*10	1355	2521	14400.0	Feasible	1991	0.8	-21.02
CBJSS-LA31	30*10	1784	3527	21600.0	Feasible	2852	2.9	-19.14
CBJSS-LA32	30*10	1850	3673	21600.0	Feasible	3049	2.6	-16.99
CBJSS-LA33	30*10	1719	3583	21600.0	Feasible	2749	2.5	-23.28
CBJSS-LA34	30*10	1721	3625	21600.0	Feasible	2909	2.6	-19.75
CBJSS-LA35	30*10	1888	3742	21600.0	Feasible	2966	2.8	-20.74
CBJSS-LA36	15*15	1028	2463	21600.0	Feasible	1803	5.5	-26.80
CBJSS-LA37	15*15	986	2654	21600.0	Feasible	1928	5.9	-27.35
CBJSS-LA38	15*15	1171	2426	21600.0	Feasible	1757	5.9	-27.58
CBJSS-LA39	15*15	1012	2544	21600.0	Feasible	1825	5.7	-28.26
CBJSS-LA40	15*15	1222	2411	21600.0	Feasible	1750	5.7	-27.42

^aThe lower bound of each CBJSS instance is computed by $LB = \max\left\{\max_i \sum_{j=1}^n p_{ij}, \max_j \sum_{i=1}^m p_{ij}\right\}$, where p_{ij} is the processing time of job j by machine i .

^bThe solution status of ILOG-CPLEX is indicated by the ILOG function, i.e. *Cplex.GetStatus()*.

BJSS_LA01 and NWJSS_LA01 instances shows that the timing of operations are distinct in three types of scheduling systems (i.e. the CBJSS, BJSS and NWJSS systems), due to different configurations of buffering requirements among machines. By comparing three makespan values (i.e. 1181, 1439 and 1618), it is validated that the CBJSS_LA01 schedule is much better than the corresponding BJSS_LA01 and NWJSS_LA01 schedules. In real-world implementation, the CBJSS system is usually more flexible than the BJSS and NWJSS systems, implying that the CBJSS system can obtain the better schedule with much less computational efforts.

Computational results on 40 benchmark *CBJSS-LA* instances are summarised in Table 1, which is structured as follows. The names of *CBJSS* instances are shown in the first column. The second column ($n*m$) represents the problem size of each instance. The third column (LB) shows the lower bound value on the makespan of each *CBJSS-LA* instance, which is also the lower bound of each corresponding *JSS-LA* instance (Liu and Kozan 2012a). The fourth column (*Makespan1*) presents the solutions obtained by the MIP solver (i.e. IBM ILOG-CPLEX 12.4). The fifth column (*Time1*)

Table 2. Computational results of other large size benchmark CBJSS instances.

Instances	$n*m$	LB	ILOG-CPLEX			CBJSS-CA-BIH Algorithm		
			Makespan1	Time1 (s)	Status	Makespan2	Time2 (s)	Gap (%)
swv01_CBJSS	20*10	1219	2489	10800.0	Feasbile	2164	4.35	-13.06
swv02_CBJSS	20*10	1259	2625	10800.0	Feasbile	2083	4.21	-20.65
swv03_CBJSS	20*10	1178	2544	10800.0	Feasbile	2296	4.53	-9.75
swv04_CBJSS	20*10	1161	2729	10800.0	Feasbile	2291	4.81	-16.05
swv05_CBJSS	20*10	1235	2771	10800.0	Feasbile	2373	4.62	-14.36
swv06_CBJSS	20*15	1229	3416	14400.0	Feasbile	2801	8.19	-18.00
swv07_CBJSS	20*15	1128	3311	14400.0	Feasbile	2858	8.95	-13.68
swv08_CBJSS	20*15	1330	3700	14400.0	Feasbile	2719	8.24	-26.51
swv09_CBJSS	20*15	1266	3082	14400.0	Feasbile	2692	8.82	-12.65
swv10_CBJSS	20*15	1159	3246	14400.0	Feasbile	2820	8.36	-13.12
swv11_CBJSS	50*10	2808	6075	21600.0	Feasbile	4855	210.61	-20.08
swv12_CBJSS	50*10	2829	6229	21600.0	Feasbile	4840	200.74	-22.30
swv13_CBJSS	50*10	2977	6544	21600.0	Feasbile	4885	220.18	-25.35
swv14_CBJSS	50*10	2842	5607	21600.0	Feasbile	4563	199.78	-18.62
swv15_CBJSS	50*10	2762	5699	21600.0	Feasbile	4634	194.45	-18.69
swv16_CBJSS	50*10	2924	5544	21600.0	Feasbile	4919	247.79	-11.27
swv17_CBJSS	50*10	2794	5760	21600.0	Feasbile	4727	215.61	-17.93
swv18_CBJSS	50*10	2852	5595	21600.0	Feasbile	4715	218.39	-15.73
swv19_CBJSS	50*10	2843	5605	21600.0	Feasbile	4799	224.87	-14.38
swv20_CBJSS	50*10	2823	5230	21600.0	Feasbile	4827	231.75	-7.71
yn1_CBJSS	20*20	694	1881	21600.0	Feasbile	1731	14.75	-7.97
yn2_CBJSS	20*20	713	1974	21600.0	Feasbile	1752	14.46	-11.25
yn3_CBJSS	20*20	680	2105	21600.0	Feasbile	1744	15.12	-17.15
yn4_CBJSS	20*20	719	2076	21600.0	Feasbile	1817	14.28	-12.48
Average								-15.78

presents the CPU times of ILOG-CPLEX, measured in seconds. The solution status (*Status*) in the sixth column indicates if an instance is optimally solved by ILOG-CPLEX. In the same fashion, the seventh column (*Makespan2*) shows the solutions by the *CBJSS-CA-BIH* algorithm and their CPU times are given in the eighth column (*Time2*). The last column (*Gap*), which is defined by $(Makespan2 - Makespan1) \times 100 / Makespan1$, shows the corresponding deviation percentage away from the ILOG-CPLEX solution. Based on the results in Table 1, one finding is that the proposed *CBJSS-CA-BIH* algorithm can obtain the near-optimal solution of small size instances (e.g. *CBJSS-LA01-10* instances) with an average optimality gap of less than 4%. In evaluation of computational times, the *CBJSS-CA-BIH* algorithm is much more efficient as it takes much less CPU time than that of ILOG-CPLEX. For example, for instance, *CBJSS_LA05*, ILOG-CPLEX spends 94.2 s to find the exact solution with the makespan of 599; in comparison, the *CBJSS-CA-BIH* algorithm finds the solution with the makespan of 603 but takes less than 0.1 s. Another observation in Table 1 is that ILOG-CPLEX fails to solve large size instances in an acceptable running time or due to memory overflow. In real-life implementations, the proposed *CBJSS-CA-BIH* algorithm is more satisfactory to solve industry-scale instances in real time.

More large-scale benchmark CBJSS instances are established based on swv01-swv20 data and yn1-yn4 data provided in the OR-Lib (Beasley 1990). Due to computational complexity, the optimal (or even good enough) results of large size CBJSS instances cannot be achieved by the exact MIP solver in a given reasonable time (even with the time limit of up to 6 h). According to our analysis, one of main reasons is mostly due to the memory limit of the academic version (i.e. IBM ILOG-CPLEX V12.4 for academic use in this study). As shown in Table 2, in comparison to the exact MIP solver, the proposed *CBJSS-CA-BIH* algorithm can find the better (15.78% improvement on average) solutions for large-scale CBJSS instances with much less computational times.

To evaluate the performance of the CBJSS system, the obtained CBJSS solutions of 40 Lawrence benchmark instances (i.e. *CBJSS_LA01* to *CBJSS_LA40*) are compared with the optimal solutions of their corresponding JSS instances (i.e. *JSS_LA01* to *JSS_LA40*) (Liu and Ong 2004) and BJSS instances (i.e. *BJSS_LA01* to *BJSS_LA40*) given in the literature (Gröflin and Klinkert 2009). As shown in the first three columns in Table 3, the optimal JSS solutions, the optimal BJSS solutions and the obtained CBJSS solution are, respectively, denoted Opt_{JSS} , Opt_{BJSS} and Sol_{CBJSS} . In the forth column, Comparison 1 shows that the average deviation between Opt_{JSS} and Opt_{BJSS} is 46.82%.

Table 3. Comparison among the results of Lawrence JSS, BJSS and CBJSS benchmark instances.

Opt _{JSS}	Opt _{BJSS}	Sol _{CBJSS}	Comparison 1 (Opt _{BJSS} /Opt _{JSS} - 1) ×100%	Comparison 2 (Sol _{CBJSS} /Opt _{JSS} - 1) ×100%	Comparison 3 (Opt _{BJSS} /Sol _{CBJSS} - 1) ×100%
666	832	718	24.92	7.81	15.88
655	793	729	21.07	11.30	8.78
597	747	701	25.13	17.42	6.56
590	769	657	30.34	11.36	17.05
593	698	603	17.71	1.69	15.75
926	1180	1070	27.43	15.55	10.28
890	1091	1000	22.58	12.36	9.10
863	1125	1027	30.36	19.00	9.54
951	1223	1119	28.60	17.67	9.29
958	1203	1104	25.57	15.24	8.97
1222	1584	1484	29.62	21.44	6.74
1039	1391	1325	33.88	27.53	4.98
1150	1548	1413	34.61	22.87	9.55
1292	1620	1475	25.39	14.16	9.83
1207	1650	1512	36.70	25.27	9.13
945	1142	1023	20.85	8.25	11.63
784	1026	920	30.87	17.35	11.52
848	1078	964	27.12	13.68	11.83
842	1093	935	29.81	11.05	16.90
902	1154	1005	27.94	11.42	14.83
1048	1545	1294	47.42	23.47	19.40
927	1458	1158	57.28	24.92	25.91
1032	1611	1453	56.10	40.79	10.87
935	1571	1393	68.02	48.98	12.78
977	1499	1411	53.43	44.42	6.24
1218	2162	1939	77.50	59.20	11.50
1242	2175	1918	75.12	54.43	13.40
1216	2071	1916	70.31	57.57	8.09
1182	2124	1817	79.70	53.72	16.90
1355	2171	1991	60.22	46.94	9.04
1784	3167	2852	77.52	59.87	11.04
1850	3418	3049	84.76	64.81	12.10
1719	3131	2749	82.14	59.92	13.90
1721	3205	2909	86.23	69.03	10.18
1888	3311	2966	75.37	57.10	11.63
1268	1932	1803	52.37	42.19	7.15
1397	2053	1928	46.96	38.01	6.48
1203	1875	1757	55.86	46.05	6.72
1233	1950	1825	58.15	48.01	6.85
1228	1936	1750	57.65	42.51	10.63
Average			46.82	32.11	11.22

In the forth column, Comparison 2 shows that the average deviation between Opt_{JSS} and Sol_{CBJSS} is 32.11%. In the last column, the direct comparison between Opt_{BJSS} and Sol_{CBJSS} indicates that the CBJSS solutions lead to productivity improvement by 11.22%. Three comparisons quantitatively validate that the CBJSS system is much more efficient than the BJSS system due to its flexibility of buffering resources and diversification of in-process buffering constraints (Table 3).

6. Conclusions

In this paper, we investigate a generalised job shop with a combination of four buffering constraints (i.e. *no-wait*, *no-buffer*, *limited-buffer* and *infinite-buffer*) and then define a new scheduling problem called the *CBJSS*. Its critical problem properties are thoroughly analysed in terms of the Gantt charts. Based on the analysis of problem properties, the *CBJSS* is mathematically formulated and a fast best-insertion-heuristic algorithm embedded with an innovative constructive

algorithm is developed to solve *CBJSS* in an efficient way. Computational results based on a collection of established *CBJSS* benchmark instances show that the proposed solution approach is satisfactory to solve industry-scale instances. A comparative study shows that the *CBJSS* can beat the current state of the art in its specific problems (e.g. the *BJSS*), which implies that the *CBJSS* is flexible to establish a more productive scheduling system. In real-world implementation, the *CBJSS* methodology is promising to be applied as a fundamental tool to analyse, model, solve and evaluate many complex industrial systems that must consider real-life buffering constraints (Che et al. 2011, 2012; Kozan and Liu 2012; Liu and Kozan 2012b, forthcoming; Yan et al. 2012; Gholami, Sotskov, and Werner 2013; Viergutz and Knust 2014; Kozan and Liu 2016; Madaan, Chan, and Niu 2016; Zhan et al. 2016; Eltokhy, Chan, and Chung 2017; Kozan and Liu 2017; Liu and Kozan 2017; Samà, Pellegrini et al. 2017; Yan et al., forthcoming). For example, the basic train scheduling problem can be transformed into a job shop scheduling system with blocking (hold-while-wait) constraints because a railway network consists of single-track sections, sidings, crossing loops and stations. Please see a detailed comparison of approaches to modelling and solving train scheduling problems as job shops with blocking constraints, recently discussed by Lange and Werner (forthcoming). Another recent real-life verification comes from the application of flexible job shop scheduling with blocking and no-wait constraints to scheduling health care activities in an Australian hospital by Burdett and Kozan (2017). Regarding the scope for future research, the proposed *CBJSS* methodology will be implemented in our industry-link robotic cell, healthcare, aviation and open-pit mining projects.

Acknowledgement

The authors would like to acknowledge the partial support from the Cooperative Research Centre for Optimising Resource Extraction established by the Australian Government's Cooperative Research Centres Programme as well as the National Natural Science Foundation of China under grant numbers. 71531003, 71571032 and 71572156.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This work was partially supported from the Cooperative Research Centre for Optimising Resource Extraction established by the Australian Government's Cooperative Research Centres Programme as well as the National Natural Science Foundation of China [grant number 71531003], [grant number 71571032], [grant number 71572156].

ORCID

Erhan Kozan  <http://orcid.org/0000-0002-3208-702X>

References

- Amirghasemi, M., and R. Zamani. 2015. "An Effective Asexual Genetic Algorithm for Solving the Job Shop Scheduling Problem." *Computers & Industrial Engineering* 83: 123–138. doi:10.1016/j.cie.2015.02.011.
- Bai, D., J. Liang, B. Liu, M. Tang, and Z. Zhang. 2017. "Permutation Flow Shop Scheduling Problem to Minimize Nonlinear Objective Function with Release Dates." *Computers & Industrial Engineering* 112: 336–347.
- Beasley, J. E. 1990. "OR-Library: Distributing Test Problems by Electronic Mail." *Journal of the Operational Research Society* 41: 1069–1072. doi:10.1057/jors.1990.166.
- Brucker, P., S. Heitmann, J. Hurink, and T. Nieberg. 2006. "Job-shop Scheduling with Limited Capacity Buffers." *OR Spectrum* 28: 151–176. doi:10.1007/s00291-005-0008-1.
- Brucker, P., and T. Kampmeyer. 2008. "Cyclic Job Shop Scheduling Problems with Blocking." *Annals of Operations Research* 159: 161–181. doi:10.1007/s10479-007-0276-z.
- Brucker, P., and S. Knust. 2012. "Chapter 4.7 Job-Shop Problems with Limited Buffers." In *Complex Sched.* 2nd ed., 300–315. Heidelberg: Springer.
- Burdett, R. L., and E. Kozan. 2009. "Techniques for Inserting Additional Trains into Existing Timetables." *Transportation Research Part B: Methodological* 43: 821–836. doi:10.1016/j.trb.2009.02.005.
- Burdett, R. L., and E. Kozan. 2010. "A Disjunctive Graph Model and Framework for Constructing New Train Schedules." *European Journal of Operational Research* 200: 85–98. doi:10.1016/j.ejor.2008.12.005.
- Burdett, R., and E. Kozan. 2017. "An Integrated Approach for Scheduling Health Care Activities in a Hospital." *European Journal of Operational Research* 264: 756–773.

- Bürgy, R., and H. Gröflin. 2016. "The Blocking Job Shop with Rail-bound Transportation." *Journal of Combinatorial Optimization* 31: 152–181. doi:10.1007/s10878-014-9723-3.
- Carlier, J., and E. Pinson. 1989. "An Algorithm for Solving the Job-shop Problem." *Management Science* 35: 164–176. doi:10.1287/mnsc.35.2.164.
- Chan, F. T. S., and K. L. Choy. 2011. "A Genetic Algorithm-based Scheduler for Multiproduct Parallel Machine Sheet Metal Job Shop." *Expert Systems with Applications* 38: 8703–8715. doi:10.1016/j.eswa.2011.01.078.
- Che, A., M. Chabrol, M. Gourgand, and Y. Wang. 2012. "Scheduling Multiple Robots in a No-wait Re-Entrant Robotic Flowshop." *International Journal of Production Economics* 135: 199–208. doi:10.1016/j.ijpe.2011.07.008.
- Che, A., H. Hu, M. Chabrol, and M. Gourgand. 2011. "A Polynomial Algorithm for Multi-Robot 2-cyclic Scheduling in a No-wait Robotic Cell." *Computers & Operations Research* 38: 1275–1285. doi:10.1016/j.cor.2010.11.008.
- Chien, C. F., F. P. Tseng, and C. H. Chen. 2008. "An Evolutionary Approach to Rehabilitation Patient Scheduling: A Case Study." *European Journal of Operational Research* 189: 1234–1253. doi:10.1016/j.ejor.2007.01.062.
- D'Ariano, A., D. Pacciarelli, and M. Pranzo. 2007. "A Branch and Bound Algorithm for Scheduling Trains in a Railway Network." *European Journal of Operational Research* 183: 643–657. doi:10.1016/j.ejor.2006.10.034.
- D'Ariano, A., D. Pacciarelli, and M. Pranzo. 2008. "Assessment of Flexible Timetables in Real-time Traffic Management of a Railway Bottleneck." *Transportation Research Part C: Emerging Technologies* 16: 232–245. doi:10.1016/j.trc.2007.07.006.
- Davendra, D., and M. Bialic-Davendra. 2013. "Scheduling Flow Shops with Blocking Using a Discrete Self-organising Migrating Algorithm." *International Journal of Production Research* 51: 2200–2218. doi:10.1080/00207543.2012.711968.
- Ding, J., S. Song, J. N. D. Gupta, C. Wang, and R. Zhang. 2015. "New Block Properties for Flowshop Scheduling with Blocking and Their Application in an Iterated Greedy Algorithm." *International Journal of Production Research* 54: 4759–4772. doi:10.1080/00207543.2015.1076941.
- Eltoukhy, A. E. E., F. T. S. Chan, and S. H. Chung. 2017. "Airline Schedule Planning: A Review and Future Directions." *Industrial Management & Data Systems* 117: 1201–1243. doi:10.1108/IMDS-09-2016-0358.
- Fink, A., and S. Voß. 2003. "Solving the Continuous Flow-shop Scheduling Problem by Metaheuristics." *European Journal of Operational Research* 151: 400–414. doi:10.1016/S0377-2217(02)00834-2.
- Fisher, H., and G. L. Thompson. 1963. "Industrial Scheduling." In *Industrial Scheduling*, edited by J. F. Muth and G. L. Thompson, 225–251. Englewood Cliffs, NJ: Prentice Hall.
- Fu, Q., A. I. Sivakumar, and K. Li. 2012. "Optimisation of Flow-shop Scheduling with Batch Processor and Limited Buffer." *International Journal of Production Research* 50: 2267–2285. doi:10.1080/00207543.2011.565813.
- Gholami, O., Y. N. Sotskov, and F. Werner. 2013. "Fast Edge-orientation Heuristics for Job-shop Scheduling Problems with Applications to Train Scheduling." *International Journal of Operational Research* 2: 19–32.
- Grabowski, J., and J. Pempera. 2007. "The Permutation Flow Shop Problem with Blocking. A Tabu Search Approach." *Omega* 35: 302–311. doi:10.1016/j.omega.2005.07.004.
- Gröflin, H., and A. Klinkert. 2006. "Feasible Insertions in Job Shop Scheduling, Short Cycles and Stable Sets." *European Journal of Operational Research* 177: 763–785. doi:10.1016/j.ejor.2005.12.025.
- Gröflin, H., and A. Klinkert. 2009. "A New Neighborhood and Tabu Search for the Blocking Job Shop." *Discrete Applied Mathematics* 157: 3643–3655. doi:10.1016/j.dam.2009.02.020.
- Gröflin, H., D. N. Pham, and R. Bürgy. 2011. "The Flexible Blocking Job Shop with Transfer and Set-up times." *Journal of Combinatorial Optimization* 22: 121–144. doi:10.1007/s10878-009-9278-x.
- Hall, N. G., and C. Srisikandarajah. 1996. "A Survey of Machine Scheduling Problems with Blocking and No-wait in Process." *Operations Research* 44:510–525. doi:10.1287/opre.44.3.510.
- Han, Y., D. Gong, J. Li, and Y. Zhang. 2016. "Solving the Blocking Flow Shop Scheduling Problem with Makespan Using a Modified Fruit Fly Optimisation Algorithm." *International Journal of Production Research* 54: 6782–6797. doi:10.1080/00207543.2016.1177671.
- Hauptman, B., and V. Jovan. 2004. "An Approach to Process Production Reactive Scheduling." *ISA Transactions* 43: 305–318. doi:10.1016/S0019-0578(07)60039-1.
- Kozan, E., and S. Q. Liu. 2012. "A Demand-responsive Decision Support System for Coal Transportation." *Decision Support Systems* 54: 665–680. doi:10.1016/j.dss.2012.08.012.
- Kozan, E., and S. Q. Liu. 2016. "A New Open-pit Multi-stage Mine Production Timetabling Model for Drilling, Blasting and Excavating Operations." *Mining Technology* 125: 47–53. doi:10.1179/1743286315Y.0000000031.
- Kozan, E., and S. Q. Liu. 2017. "An Operational-level Multi-stage Mine Production Timetabling Model for Optimally Synchronising Drilling, Blasting and Excavating Operations." *International Journal of Mining, Reclamation and Environment* 31: 457–474. doi:10.1080/17480930.2016.1160818.
- Lange, J., and F. Werner. Forthcoming. "Approaches to Modeling Train Scheduling Problems as Job-Shop Problems with Blocking Constraints." *Journal of Scheduling*. doi:10.1007/s10951-017-0526-0.
- Leisten, R. 1990. "Flowshop Sequencing Problems with Limited Buffer Storage." *International Journal of Production Research* 28: 2085–2100. doi:10.1080/00207549008942855.
- Liu, S. Q., and E. Kozan. 2009a. "Scheduling a Flow Shop with Combined Buffer Conditions." *International Journal of Production Economics* 117: 371–380. doi:10.1016/j.ijpe.2008.11.007.

- Liu, S. Q., and E. Kozan. 2009b. "Scheduling Trains as a Blocking Parallel-machine Job Shop Scheduling Problem." *Computers & Operations Research* 36: 2840–2852. doi:10.1016/j.cor.2008.12.012.
- Liu, S. Q., and E. Kozan. 2011a. "Optimising a Coal Rail Network under Capacity Constraints." *Flexible Services and Manufacturing Journal* 23: 90–110. doi:10.1007/s10696-010-9069-9.
- Liu, S. Q., and E. Kozan. 2011b. "Scheduling Trains with Priorities: A No-wait Blocking Parallel-machine Job-shop Scheduling Model." *Transportation Science* 45: 175–198. doi:10.1287/trsc.1100.0332.
- Liu, S. Q., and E. Kozan. 2012a. "A Hybrid Shifting Bottleneck Procedure Algorithm for the Parallel-machine Job-Shop Scheduling Problem." *Journal of the Operational Research Society* 63: 168–182. doi:10.1057/jors.2011.4.
- Liu, S. Q., and E. Kozan. 2012b. "An Interactive Planning and Scheduling Framework for Optimising Pits-to-Crushers Operations." *Industrial Engineering & Management Systems* 11: 94–102. doi:10.7232/iems.2012.11.1.094.
- Liu, S. Q., and E. Kozan. 2017. "A Hybrid Metaheuristic Algorithm to Optimise a Real-world Robotic Cell." *Computers & Operations Research* 84: 188–194. doi:10.1016/j.cor.2016.09.011.
- Liu, S. Q., and E. Kozan. Forthcoming. "Integration of Mathematical Models for Ore Mining Industry." *International Journal of Systems Science: Operations & Logistics*. doi:10.1080/23302674.2017.1344330.
- Liu, S. Q., and H. L. Ong. 2002. "A Comparative Study of Algorithms for the Flowshop Scheduling Problem." *Asia-Pacific Journal of Operational Research* 19: 205–222. doi:10.1017/CBO9781107415324.004.
- Liu, S. Q., and H. L. Ong. 2004. "Metaheuristics for the Mixed Shop Scheduling Problem." *Asia-Pacific Journal of Operational Research* 21: 97–115.
- Liu, S. Q., H. L. Ong, and K. M. Ng. 2005. "Metaheuristics for Minimizing the Makespan of the Dynamic Shop Scheduling Problem." *Advances in Engineering Software* 36: 199–205. doi:10.1016/j.advengsoft.2004.10.002.
- Louaqad, S., and O. Kamach. 2016. "Mixed Integer Linear Programs for Blocking and No Wait Job Shop Scheduling Problems in Robotic Cells." *International Journal of Computer Applications* 153: 1–7.
- Madaan, J., F. T. S. Chan, and B. D. Niu. 2016. "Strategies for Evaluating Performance of Flexibility in Product Recovery System." *International Journal of Production Research* 54: 2895–2906. doi:10.1080/00207543.2015.1120899.
- Mascis, A., and D. Pacciarelli. 2002. "Job-shop Scheduling with Blocking and No-wait Constraints." *European Journal of Operational Research* 143: 498–517. doi:10.1016/S0377-2217(01)00338-1.
- Masoud, M., G. Kent, E. Kozan, and S. Q. Liu. 2016. "A New Multi-objective Model to Optimise Rail Transport Scheduler." *Journal of Transportation Technologies* 06: 86–98.
- Masoud, M., E. Kozan, and G. Kent. 2011. "A Job-shop Scheduling Approach for Optimising Sugarcane Rail Operations." *Flexible Services and Manufacturing Journal* 23: 181–206.
- Masoud, M., E. Kozan, and G. Kent. 2015. "Hybrid Metaheuristic Techniques for Optimising Sugarcane Rail Operations." *International Journal of Production Research* 53: 2569–2589.
- Masoud, M., E. Kozan, G. Kent, and S. Q. Liu. 2016. "An Integrated Approach to Optimise Sugarcane Rail Operations." *Computers & Industrial Engineering* 98: 211–220. doi:10.1016/j.cie.2016.06.002.
- Masoud, M., E. Kozan, G. Kent, and S. Q. Liu. 2017. "A New Constraint Programming Approach for Optimising a Coal Rail System." *Optimization Letters* 11: 725–738. doi:10.1007/s11590-016-1041-5.
- Mati, Y., C. Lahlou, and S. Dauzère-Pères. 2011. "Modelling and Solving a Practical Flexible Job-shop Scheduling Problem with Blocking Constraints." *International Journal of Production Research* 49: 2169–2182. doi:10.1080/00207541003733775.
- Mati, Y., N. Rezg, and X. Xie. 2001. "A Taboo Search Approach for Deadlock-free Scheduling of Automated Manufacturing Systems." *Journal of Intelligent Manufacturing* 12: 535–552. doi:10.1023/A:1012260622596.
- Nawaz, M., E. E. Enscore, and I. Ham. 1983. "A Heuristic Algorithm for the M-machine, N-job Flow-shop Sequencing Problem." *Omega* 11: 91–95. doi:10.1016/0305-0483(83)90088-9.
- Pacciarelli, D. 2002. "Alternative Graph Formulation for Solving Complex Factory-scheduling Problems." *International Journal of Production Research* 40: 3641–3653.
- Peng, B., Z. Lü, and T. C. E. Cheng. 2015. "A Tabu Search/Path Relinking Algorithm to Solve the Job Shop Scheduling Problem." *Computers & Operations Research* 53: 154–164. doi:10.1016/j.cor.2014.08.006.
- Pham, D.-N., and A. Klinkert. 2008. "Surgical Case Scheduling as a Generalized Job Shop Scheduling Problem." *European Journal of Operational Research* 185: 1011–1025. doi:10.1016/j.ejor.2006.03.059.
- Pranzo, M., and D. Pacciarelli. 2016. "An Iterated Greedy Metaheuristic for the Blocking Job Shop Scheduling Problem." *Journal of Heuristics* 22: 587–611. doi:10.1007/s10732-014-9279-5.
- Qian, B., L. Wang, D. X. Huang, and X. Wang. 2009. "An Effective Hybrid DE-based Algorithm for Flow Shop Scheduling with Limited Buffers." *International Journal of Production Research* 47: 1–24. doi:10.1080/00207540701528750.
- Ronconi, D. P. 2004. "A Note on Constructive Heuristics for the Flowshop Problem with Blocking." *International Journal of Production Economics* 87: 39–48. doi:10.1016/S0925-5273(03)00065-3.
- Rossit, D. A., F. Tohmé, and M. Frutos. Forthcoming. "The Non-permutation Flow-shop Scheduling Problem: A Literature Review." *Omega*. doi:10.1016/j.omega.2017.05.010.
- Ruan, J. H., X. P. Wang, F. T. S. Chan, and Y. Shi. 2016. "Optimizing the Intermodal Transportation of Emergency Medical Supplies using Balanced Fuzzy Clustering." *International Journal of Production Research* 54: 4368–4386. doi:10.1080/00207543.2016.1174344.

- Samà, M., A. D'Ariano, P. D'Ariano, and D. Pacciarelli. 2017. "Scheduling Models for Optimal Aircraft Traffic Control at Busy Airports: Tardiness, Priorities, Equity and Violations Considerations." *Omega* 67: 81–98.
- Samà, M., P. Pellegrini, A. D'Ariano, J. Rodriguez, and D. Pacciarelli. 2017. "On the Tactical and Operational Train Routing Selection Problem." *Transportation Research Part C: Emerging Technologies* 76: 1–15. doi:10.1016/j.trc.2016.12.010.
- Samarghandi, H., and T. Y. ElMekkawy. 2013. "Two-machine, No-wait Job Shop Problem with Separable Setup times and Single-server Constraints." *The International Journal of Advanced Manufacturing Technology* 65: 295–308. doi:10.1007/s00170-012-4169-1.
- Santosa, B., M. A. Budiman, and S. E. Wiratno. 2011. "A Cross Entropy-genetic Algorithm for M-machines No-wait Job-shop Scheduling Problem." *Journal of Intelligent Learning Systems and Applications* 3: 171–180. doi:10.4236/jilsa.2011.33018.
- Schuster, C. J., and J. M. Framinan. 2003. "Approximative Procedures for No-wait Job Shop Scheduling." *Operations Research Letters* 31: 308–318. doi:10.1016/S0167-6377(03)00005-1.
- Song, J., and T. Lee. 1998. "Petri Net Modeling and Scheduling for Cyclic Job Shops with Blocking." *Computers & Industrial Engineering* 34: 281–295. doi:10.1016/S0360-8352(97)00325-2.
- Toba, H. 2005. "A Tight Flow Control for Job-shop Fabrication Lines with Finite Buffers." *IEEE Transactions on Automation Science and Engineering* 2: 78–83. doi:10.1109/TASE.2004.829440.
- Viergutz, C., and S. Knust. 2014. "Integrated Production and Distribution Scheduling with Lifespan Constraints." *Annals of Operations Research* 213: 293–318. doi:10.1007/s10479-012-1197-z.
- Witt, A., and S. Voß. 2007. "Simple Heuristics for Scheduling with Limited Intermediate Storage." *Computers & Operations Research* 34: 2293–2309. doi:10.1016/j.cor.2005.09.004.
- Yan, P., A. Che, E. Levner, and S. Q. Liu. Forthcoming. "A Heuristic for Inserting Randomly Arriving Jobs into an Existing Hoist Schedule." *IEEE Transactions on Automation Science and Engineering*. doi:10.1109/TASE.2017.2749429.
- Yan, P., A. Che, N. Yang, and C. Chu. 2012. "A Tabu Search Algorithm with Solution Space Partition and Repairing Procedure for Cyclic Robotic Cell Scheduling Problem." *International Journal of Production Research* 50: 6403–6418. doi:10.1080/00207543.2011.645953.
- Zeng, C., J. Tang, and C. Yan. 2014. "Scheduling of No Buffer Job Shop Cells with Blocking Constraints and Automated Guided Vehicles." *Applied Soft Computing* 24: 1033–1046. doi:10.1016/j.asoc.2014.08.028.
- Zhan, S., L. G. Kroon, J. Zhao, and Q. Peng. 2016. "A Rolling Horizon Approach to the High Speed Train Rescheduling Problem in Case of a Partial Segment Blockage." *Transportation Research Part E: Logistics and Transportation Review* 95: 32–61. doi:10.1016/j.tre.2016.07.015.
- Zhang, R., and R. Chiong. 2016. "Solving the Energy-efficient Job Shop Scheduling Problem: A Multi-objective Genetic Algorithm with Enhanced Local Search for Minimizing the Total Weighted Tardiness and Total Energy Consumption." *Journal of Cleaner Production* 112: 3361–3375. doi:10.1016/j.jclepro.2015.09.097.
- Zhang, J., G. Ding, Y. Zou, S. Qin, and J. Fu. Forthcoming. "Review of Job Shop Scheduling Research and Its New Perspectives under Industry 4.0." *Journal of Intelligent Manufacturing*. doi:10.1007/s10845-017-1350-2.
- Zhang, C., Z. Shi, Z. Huang, Y. Wu, and L. Shi. 2017. "Flow Shop Scheduling with a Batch Processor and Limited Buffer." *International Journal of Production Research* 55: 3217–3233. doi:10.1080/00207543.2016.1268730.