Scheduling of a job-shop problem with limited output buffers

Chengkuan Zeng\textsuperscript{a}, Jiafu Tang\textsuperscript{b}, Zhi-Ping Fan\textsuperscript{a} and Chongjun Yan\textsuperscript{b}

\textsuperscript{a}Department of Information Management and Decision Sciences, School of Business Administration, Northeastern University, Shenyang, People's Republic of China; \textsuperscript{b}College of Management Science \\& Engineering, Dongbei University of Finance and Economics (DUFE), Dalian, People's Republic of China

**ABSTRACT**

This article addresses a job-shop problem with limited output buffers (JS-LOB) with the objective of minimizing the process makespan. An integer nonlinear mathematical programming model is proposed to describe this problem. Based on the model, a two-stage algorithm consisting of obtaining feasible solutions and a local search is proposed to solve the JS-LOB problem. The local search has two operators: the first is a neighbourhood structure based on a disjunctive graph model, and the second is similar to crossover in the genetic algorithm to avoid falling into local optima. Computational results are presented for a set of benchmark tests. The results show the effectiveness of the proposed algorithm and indicate whether the processing time of the job conforms to a uniform distribution. When the proportion between the capacity of the buffer and the number of jobs is larger than 20%, the influence of the buffer becomes very small.

**ARTICLE HISTORY**

Received 2 May 2018
Accepted 18 January 2019

**KEYWORDS**

Job-shop scheduling; limited output buffer; disjunctive graph model; uniform distribution

---

1. Introduction

Job-shop problems (JSPs) are some of the best known scheduling problems (Brucker \textit{et al.} 2006). In a classical JSP, to minimize the makespan for each job, a specific route through the machines is defined (Brucker \textit{et al.} 2006). With developments in manufacturing, extended JSPs have appeared. Additional elements under consideration include transportation (Zeng, Tang, and Yan 2014), set-up time (Sun 2009), due date or tardiness (Tang, Zeng, and Pan 2016), machine breakdown or maintenance (Zeng, Tang, and Fan 2018), limited buffers (Brucker \textit{et al.} 2006), available job dynamic arrival (Zeng \textit{et al.} 2018) and random processing time (Tokola, Ahlroth, and Niemi 2014).

The buffer is the space in which jobs are allowed to wait on a machine if the next machine is not available (Brucker \textit{et al.} 2006). With the increase in just-in-time manufacturing systems, which maintain a limited process inventory (Pan, Wang, and Gao 2011), studies on scheduling problems with limited buffer space are attracting more attention. According to the capacity of the buffer, when the capacity is zero, it will exhibit either blocking or a no-wait constraint, depending on whether jobs waiting on the machines are allowed (Mascis and Pacciarelli 2002). When the capacity is a positive number, after the job finishes processing on the current machine, if the buffer is not completely occupied, the job can move to the buffer and release the current machine; otherwise, the job has to wait on...
the current machine until the next machine is available or a free place is available in the buffer, which causes blocking. Because of the complexity of scheduling under a limited buffer constraint, most of the existing studies are about flow shops (Pan et al. 2011), and JSPs with limited buffer space have only been researched by a few researchers (Brucker et al. 2006; Fahmy, ElMekkawy, and Balakrishnan 2008; Witt and Voß 2010; Liu et al. 2017). Brucker et al. (2006) found a compact representation of the solution for a JSP with limited buffer space, especially for a general buffer. In Brucker et al. (2006), the following types of buffers are defined: if any assignment of operations to the buffer is possible, it is called a general buffer; if the assignment depends on the job index, which means that each job has (or does not have) its own buffer, it is called a job-dependent buffer. In these two conditions, the buffer can be treated as a facility independent of machines. If the buffer is associated with each pair \((M_k, M_l)\) of machines \(M_k\) and \(M_l\), it is called a pairwise buffer. After an operation is finished on \(M_k\), its next operation needs to be processed on \(M_l\) and \(M_l\) is busy, it can move to the pairwise buffer \(B_{kl}\) to wait; if the buffer is related to \(M_k\) and the operation can wait for the next machine in the buffer after finishing its processing on \(M_k\), it is called an output buffer. Similarly, if the buffer is related to \(M_k\) and the operation can wait in the buffer for processing on \(M_l\), it is called an input buffer. Fahmy, ElMekkawy, and Balakrishnan (2008) propose a novel operation insertion algorithm based on the rank matrix to address scheduling of flexible job shops with limited-capacity buffers. Witt and Voß (2010) consider the JSP with limited buffer capacities, which takes into account that different jobs consume different buffer spaces. Liu et al. (2018) deal with a job-shop system with a combination of four buffering constraints, namely no wait, no buffer, limited buffer and infinite buffer.

In the existing research on job-shop problems with limited output buffers (JS-LOB problems), only heuristic algorithms or dispatching rules are proposed, without determination or analysis of the key elements influencing scheduling of the JS-LOB, so the existing methods dealing with JS-LOB lack relevance. The motivation of the research is to find the key element, regarded as a breakthrough point, to propose a method to obtain an optimal scheduling scheme.

In this article, a JS-LOB is considered. An integer nonlinear mathematical programming (INLP) model is proposed to describe this problem, and is an effective and efficient two-stage algorithm (TSA) for solving this problem. The deadlock situation is allowed, meaning that when deadlock occurs, two (or more) operations can move to the next machine simultaneously. In the first stage of the algorithm, according to the Nawaz–Enscore–Ham (NEH) heuristic proposed by Pan et al. (2011), it is improved and combined with a buffer scheduling mechanism to find feasible solutions quickly. After that, it needs to be optimized to obtain a solution with a better objective function. In the second stage, after finding the key element influencing the scheduling of the JS-LOB problem, two local search operators are proposed to eliminate or reduce the waiting time in the buffer to obtain an optimal scheduling scheme. The operators focus on optimizing initial solutions and maintaining the diversity of the population. The first operator is a neighbourhood structure based on a disjunctive graph model, which is called N6, proposed by Balas and Vazacopoulos (1998), to optimize the initial feasible solutions to obtain solutions with a better objective function; and the second operator is similar to the crossover in the genetic algorithm to avoid falling into the local optimum. Computational results are presented for a set of benchmark tests, some of which are enlarged by different proportions between the capacity of the buffer and the number of jobs. The results show the effectiveness of the proposed algorithm and indicate that when the processing time of a job conforms to a uniform distribution, when the proportion between the capacity of the buffer and the number of jobs is larger than 20%, the influence of the buffer will become very small.

The remainder of this article is organized as follows. Section 2 gives the description and assumptions of the JS-LOB problem and describes it using an INLP model. Section 3 introduces the structure and procedure for finding feasible solutions. Section 4 introduces the structure of two operators in a local search and the procedure to obtain a final solution. Section 5 conducts a series of experimental tests, presents the computational results and shows the effectiveness of the proposed TSA. Section 6 summarizes the conclusions of the study.
2. Problem description

The JS-LOB can be described as follows. There is a set of \( n \) jobs \( J = \{1, 2, \ldots, N\} \) and a set of \( m \) machines \( M = \{1, 2, \ldots, m\} \). The processing route of each job is unique and fixed in advance. The processing time of operations is also fixed, and they are sequentially processed on \( n \) machines. At any time, each machine can process at most one job, and each job can be processed on at most one machine. The output buffer means the exiting buffer, which is tied to the machine the job is leaving. It is assumed that all machines have the same buffer capacity. When the buffer is fully occupied and the next machine is busy, the operation has to keep waiting on the current machine until the next machine or the buffer becomes available; this is called blocking. After the last operation of a job is finished, the job will leave the production system directly, so that the last operation of a job does not need a buffer. Each job is processed without pre-emption. The objective of the problem is to find a scheduling scheme with minimum makespan.

2.1. List of symbols

The following notation is adopted to describe the above problem.

**Indices:**
- \( I \) set of jobs \( (i \in I, i = 1, 2, \ldots, N) \)
- \( J \) set of operations \( (j \in J, j = 1, 2, \ldots, n) \)
- \( K \) set of machines \( (k \in K, k = 1, 2, \ldots, m) \)
- \( p \) index of the position of jobs in the buffer

**Parameters:**
- \( O_{ij} \) \( j \)th machine operation of job \( i \)
- \( P_{ij} \) processing time of \( j \)th machine operation of job \( i \)
- \( \alpha_{ijk} = \begin{cases} 1, & \text{if operation } o_{ij} \text{ requires processing on machine } k \\ 0, & \text{otherwise} \end{cases} \)

**Decision variables:**
- \( S_{ij} \) processing start time for the \( j \)th machine operation of job \( i \)
- \( C_{ij} \) processing finish time for the \( j \)th machine operation of job \( i \)
- \( IB_{ij} \) the time when operation \( O_{ij} \) leaves the current machine or enters the buffer
- \( B_k \) number of operations processed on machine \( k \) which need a buffer
- \( x_{ijkp} = \begin{cases} 1, & \text{if operation } o_{ij} \text{ is } p \text{th operation enter the buffer on machine } k \\ 0, & \text{otherwise} \end{cases} \)
- \( y_{ijkp} = \begin{cases} 1, & \text{if operation } o_{ij} \text{ is } p \text{th operation leave the buffer on machine } k \\ 0, & \text{otherwise} \end{cases} \)

2.2. Integer nonlinear programming model

Min \( \max\{C_{ij}\} \) \hspace{1cm} (1)

subject to:

\( S_{ij} + P_{ij} = C_{ij}, \forall i, j \) \hspace{1cm} (2)
\[ IB_{ij} \geq C_{ij}, \forall i, j \]  
\[ S_{ij} \geq IB_{i(j-1)}, \forall i, j = 2 \ldots n \]  
\[ \alpha_{ijk}, \alpha_{fjk}(S_{ij} - IB_{ij})(S_{fj} - IB_{ij}) \leq 0, i \neq i', j, j' = 1, 2, \ldots, n - 1, \forall k \]  
\[ \alpha_{ijk}, \alpha_{fjk}(S_{ij} - C_{ij})(S_{fj} - C_{ij}) \leq 0, i \neq i', j = n, j' = 1, 2, \ldots, n - 1, \forall k \]  
\[ \alpha_{ijk}, \alpha_{fjk}(S_{ij} - C_{ij})(S_{fj} - C_{ij}) \leq 0, i \neq i', j = n, j' = n, \forall k \]  
\[ \alpha_{i_1j_1k_1} \ldots \alpha_{i_{N+1}j_{N+1}k}\max\{IB_{i_1j_1}, IB_{i_2j_2}, \ldots, IB_{i_{N+1}j_{N+1}}\} \]
\[ - \min\{S_{i_1(j_1+1)}, S_{i_2(j_2+1)}, \ldots, S_{i_{N+1}(j_{N+1}+1)}\} \geq 0, \]
\[ i_1 \neq i_2 \ldots \neq i_{N+1}, j_1, j_2, \ldots, j_{N+1} = 1, 2, \ldots, n - 1, \forall k \]  

Equation (1) shows the objective of the problem: to find a solution to finish processing all operations as early as possible. Equation (2) shows the relations between the time when the job (operation) begins to process and finish; once an operation starts processing, it cannot be interrupted until it finishes. Equation (3) shows the relations between the time when the job finishes on the current machine and enters the buffer; after processing is finished, the job can enter the buffer on the current machine to wait for the next machine. Equation (4) shows the relations between the time when the job leaves the current machine and begins to be processed on the next machine; when the job (operation) finishes processing on the current machine and the next machine is busy, if the buffer of the current machine is not fully occupied, the job will enter the buffer to wait until the next machine is available, when the job will leave the process and release the buffer. Equations (5)–(7) show the relations of processing different operations on the same machine, representing different indices of a range of variables, indicating that no machine can process two or more operations at the same time. Equation (8) shows the time relations between the operations entering and leaving the buffer.

It is difficult to use Equation (8) to solve the model. Under the above parameters and decision variables, only when the capacity of the buffer is equal to 1 can Equation (8) be expressed as Equation (9):

\[ \alpha_{ijk}, \alpha_{fjk}(S_{i(j+1)} - IB_{ij})(S_{f(j'+1)} - IB_{ij}) \leq 0, i \neq i', j, j' = 1, 2, \ldots, n - 1, \forall k \]  

When the capacity of the buffer is larger than 1, Equation (8) can be extended to Equations (10)–(17), as follows:

\[ \sum_{i=1}^{f} \sum_{j=1}^{n-1} \alpha_{ijk} = B_k, \forall k \]  
\[ \sum_{i=1}^{f} \sum_{j=1}^{n-1} x_{ijkp} = 1, \forall k, p = 1, 2, \ldots, B_k \]  
\[ \sum_{i=1}^{f} \sum_{j=1}^{n-1} \alpha_{ijk}x_{ijkp} = 1, \forall k, p = 1, 2, \ldots, B_k \]  
\[ \sum_{i=1}^{f} \sum_{j=1}^{n-1} y_{ijkp} = 1, \forall k, p = 1, 2, \ldots, B_k \]
\[
\sum_{i=1}^{J} \sum_{j=1}^{n-1} \alpha_{ijk} y_{ijkp} = 1, \forall k, p = 1, 2, \ldots, B_k
\]  

(14)

\[
x_{ijpk} x_{ij'}(p+1)_k (IB_{ij'} - IB_{ij}) \geq 0, \forall k, p = 1, 2, \ldots, B_k - 1, i \neq i', j, j' = 1, 2, \ldots, n - 1
\]

(15)

\[
y_{ijpk} y_{ij'}(p+1)_k (S_{ij'}(j+1) - S_{ij}(j+1)) \geq 0, \forall k, p = 1, 2, \ldots, B_k - 1, i \neq i', j, j' = 1, 2, \ldots, n - 1
\]

(16)

\[
y_{ijpk} x_{ij'}(p+N)_k (IB_{ij'} - S_{ij}(j+1)) \geq 0, \text{ if } B_k > V, \forall k, p = 1, 2, \ldots, B_k - V, i \neq i', j, j' = 1, 2, \ldots, n - 1
\]

(17)

Equation (10) shows the total number of operations processed on the (same) machine that needs a buffer. Equations (11) and (12) show the relations between operation and position. Equation (11) guarantees that each operation will have a position and will have only one operation in sequence for entering the buffer. Equation (12) guarantees that the selected machine is able to process the operation. Similarly, Equations (13) and (14) also show the relations between operation and position. Equation (13) indicates that each operation will have a position and only have one operation in sequence for leaving the buffer. Equation (14) also guarantees that the selected machine is able to process the operation. If an operation moves to the next machine from the current machine directly (without waiting in the buffer), the time at which it enters the buffer is treated as the finish time on the current machine, which also equals the time at which it leaves the buffer and is the same as the processing start time on the next machine. Equations (15)–(17) show the time relations between the operations entering and leaving the buffer. They guarantee that no more than \( V \) (the capacity of the buffer) operations are waiting in the buffer at the same time.

When the scale of the instance is medium or large, it becomes difficult to solve the mathematical model, so an algorithm is proposed to address this issue.

### 3. Proposed algorithm generating initial solutions

Compared to the classical JSP, the JS-LOB problem is more difficult to solve. Until now, only a few articles have researched this topic (Brucker et al. 2006; Fahmy, ElMekkawy, and Balakrishnan 2008; Witt and Voß 2010; Liu et al. 2017). For the JS-LOB, a method does not exist to obtain a feasible solution quickly, so an effective method needs to be proposed to address this problem.

#### 3.1. Case study

To find an effective method to solve the JS-LOB problem, a simple case is used. Table 1 shows the processing routes and times for each operation.

To obtain a feasible and high-quality solution, the case is solved using LINGO according to the mathematical model above, and the capacity of each buffer is assumed to equal 1. The scheduling scheme is shown in Figure 1.

From Figure 1, it is found that if the buffer is treated as a facility (or the same as a machine), for each job, the scheduling is the same as that of the no-wait job-shop scheduling (NWJS) problem (Zhu, Li, and Wang 2009). In the NWJS problem, the key element in scheduling is to determine the processing sequence of all jobs. Thus, to find a feasible solution for the JS-LOB problem, first, the processing sequence of the jobs must be determined.

#### Table 1. Job machine load matrix.

| Processing times |
|------------------|
| Job 1            |
| M3:1             |
| M1:3             |
| M2:6             |
| Job 2            |
| M2:8             |
| M3:4             |
| M1:10            |
| Job 3            |
| M3:5             |
| M1:4             |
| M2:8             |
The NEH heuristic a well-known heuristic, proposed by Nawaz, Enscore, and Ham (1983). Pan et al. (2011) successfully applied this heuristic to solve the flow-shop problem with limited buffer with the makespan criterion. The NEH heuristic is described as follows (Pan et al. 2011).

**Step 1.** Sort the jobs according to the ascending sums of their process times. Let the obtained sequence be \( \pi = \{\pi(1), \pi(2), \ldots, \pi(n)\} \).

**Step 2.** The first two jobs of \( \pi \) are taken, and the two possible partial sequences are evaluated. Then, the best partial sequence is chosen as the current sequence.

**Step 3.** Take job \( \pi(j), j = 3, 4, \ldots, n \), and find the best partial sequence by placing it in all possible positions of jobs that have already been scheduled. Then, the best partial sequence is selected for the next generation. (If several partial sequences have the same minimum fitness value, randomly select one value as the best.) Repeat this step until all jobs are sequenced, and the final sequence of \( n \) jobs is constructed.

According to the NEH heuristic, a processing sequence and its makespan are obtained, and the makespan is treated as the benchmark. Then, the job order is randomly changed to obtain a new sequence, and the job is scheduled based on the new sequence one by one to obtain a scheduling scheme. If the makespan of the obtained scheduling scheme is smaller than the benchmark, the new sequence could represent a better solution and record it.

### 3.2. Buffer scheduling mechanism

Jobs are scheduled based on the decision sequence by the NEH heuristic. After an operation is finished, if the next machine is available, it can move to the next machine directly; if the next machine is busy and the buffer on the current machine is not completely occupied, it can move to the buffer to wait until the next machine is available, when it can leave and release the buffer; otherwise, it can just stay on the current machine, which causes blocking. When the capacity of buffer is larger than 0, the blocking operation can wait for other jobs already in buffer being released, then move to the buffer to release the machine; however, when the capacity of the buffer is equal to 0, and there exist two or more operations blocking the machine at the same time, a deadlock situation may occur. The deadlock situation can be described as follows. An operation \( O_{ij} \) is the \( j \)th operation of job \( i \) and is processed on machine \( k \), and another operation \( O_{ab} \) is the \( b \)th operation of job \( a \) and is processed on machine \( c \). If operation \( O_{i(j+1)} \) needs to process on machine \( c \) and \( O_{a(b+1)} \) needs to process on machine \( k \), the deadlock situation will appear because none of the operations can leave its current machine. In this article, when a deadlock situation occurs, the operations can be swapped simultaneously. Swapping is conducted as shown in Figure 2. The operations highlighted are deadlock operations; in Figure 2,
Figure 2. Swapping deadlock operations between the machines: (a) swapping for two blocking operations; (b) swapping for a cycle of three blocking operations; (c) swapping for a cycle of more than three blocking operations.

all possible deadlocks can be fixed. When the number of deadlock operations is three or more, which makes a cycle, swapping is performed in a similar way.

3.3. Approach of obtaining initial solutions

Based on the NEH heuristic and the buffer scheduling mechanism, the procedure to obtain a feasible solution under a given sequence is shown in Figure 3.

Step 1. Obtain a processing sequence through the NEH method, and schedule the jobs based on the obtained sequence.

Step 2. Check the status of each operation from time-point zero (marked time-point). If an (some) operation(s) is (are) finished, mark the time-point, and go to Step 3; else, go to the next time-point and repeat the steps to check the status until an operation is finished.

Step 3. Judge whether the finished operation is the last operation of a job or not; if not, go to Step 4; if yes, stop and exit; else, go back to Step 2.

Step 4. Judge whether the finished operation is on the machine or in the buffer; go to Step 5.

Step 5. If the next machine of the operation is not available and if the operation is on the machine, go to Step 6. If the operation is in the buffer, go to Step 7; else, go to Step 8.

Step 6. If the buffer of the current machine is not fully occupied, move the operation to the buffer to wait for the next machine, update the status of machine, and go back to Step 2; else, if blocking on the current machine, go back to Step 2.

Step 7. If the operation keeps waiting for the next machine in the buffer, go back to Step 2.

Step 8. Move the operation to the next machine from the current machine or the buffer, update the status of the operation and machines, and go back to Step 2.

In Step 2, the first operation of each job can be processed directly if the machine is available. If two or more operations finish at the same time and the next machine needed is the same one, randomly choose one operation to process.
4. Local search for the algorithm

4.1. Key element influencing the JS-LOB problem

As mentioned in Section 3, if the buffer is treated as a facility, the scheduling of each job is similar to that of the NWJS problem. For the NWJS problem, the key element is to determine the processing sequence and then to calculate the start process of each job one by one based on the given sequence. Differently from the NWJS problem, for the JS-LOB problem, the scheduling of operations is not just determined by the start processing time of each job, but rather, the processing sequences on each machine are not the same as the given one. The waiting time in the buffer for each operation will
vary for different scheduling schemes, and they cannot be determined in advance (before scheduling). Therefore, the waiting times in the buffer can be treated as special operations whose processing times are not fixed, and the JS-LOB problem can be treated as an NWJS problem containing those special operations, which can be adjusted to meet the scheduling of operations determined by the start processing time of jobs under the no-wait constraint. This is also the main innovation of this research, because the waiting time in the buffer is treated as a special operation; the aim is to eliminate or reduce the waiting time to obtain an optimal scheduling scheme.

Therefore, it is supposed that the length of the waiting time in the buffer can make a huge difference in scheduling. In fact, it can have a large influence on the makespan (the time to finish processing all operations). For example, the jobs listed in Table 1 are scheduled based on sequence 1–2–3, and the scheduling scheme is shown in Figure 4.

As shown in Figure 4, for all operations waiting in the buffer, O_{33} leads to the longest time on machine 1. To decrease the waiting time, attempts are made to change the processing sequence on machine 2 (because O_{33} needs to be processed on machine 2) and exchange operations O_{13} and O_{21}. The scheduling result is shown in Figure 5.

From Figures 4 and 5, in a scheduling scheme, if an operation costs substantial time in the buffer, it may influence the makespan greatly. In addition, changing the processing sequence of operations on the next machine may eliminate or reduce this influence. Thus, in the first stage, the goal of optimization is to choose the operation for which the waiting time is large and then change the processing sequence on the next machine to reduce or eliminate the waiting time with a relatively large possibility.
4.2. Proposed neighbourhood structure

To support this approach, a disjunctive graph model is applied to address the issue. Many researchers have successfully applied the disjunctive graph model to describe and solve the JSP (Zeng, Tang, and Yan 2014; Blazewicz, Pesch, and Sterna 2000). In this subsection, the disjunctive graph model will be introduced in brief. More detailed information regarding the disjunctive graph model can be found in Zeng, Tang, and Yan (2015). For example, the jobs in Table 1 can be presented using a disjunctive graph model, as in Figure 6.
To obtain a feasible solution, all undirected disjunctive edges are turned into directed conjunctive edges in the disjunctive graph model. The turn is called a complete selection. If the complete selection is acyclic, it can represent a feasible solution. Figure 7 represents the feasible solution, as shown in Figure 1.

In a conjunctive graph model, the longest path from the START node to the END node, representing the makespan, is called the critical path. The operation in the critical path is called the critical operation, and a maximal sequence of adjacent critical operations processed on the same machine is called a critical block.

**Figure 9.** Flowchart of the proposed two-stage algorithm. NEH = Nawaz–Enscore–Ham.
Table 2. Results for instances obtained by LINGO.

| Instance | Size | 0% | 10% | 20% | 30% | 50% | 80% |
|----------|------|----|-----|-----|-----|-----|-----|
| La01     | 10 × 5 | 1023 | 864 | 734 | 734 | 732 | 732 |
| La02     | 10 × 5 | 987  | 872 | 789 | 789 | 789 | 787 |
| La03     | 10 × 5 | 976  | 776 | 712 | 712 | 712 | 712 |
| La04     | 10 × 5 | 983  | 854 | 725 | 717 | 717 | 717 |
| La05     | 10 × 5 | 912  | 753 | 684 | 680 | 680 | 680 |

Figure 10. Results obtained by the two-stage algorithm (TSA) and four compared algorithms for scale of instance 10 × 5. HA = heuristic algorithm; NOIA = novel operations insertion algorithm; GS = generation scheme; BIH = best insertion heuristic.

For the JS-LOB problem, the solution space is a subset of the classical JSP. For the critical path, only altering the location of the operations on the path can make the path shorter; otherwise, the existing critical path is fixed, and there is no way to reduce the makespan. According to the idea mentioned in Section 4.1, the main procedure of a neighbourhood is to find the critical path of the initial solution. The operation with the longest waiting time in the buffer is ensured to be the next operation. If a critical block exists before the selected operation on the machine (such as in Figure 5), then the sequence of operations in the block is exchanged and all operations are rescheduled based on the new sequence. Here, the neighbourhood structure named N6 proposed by Balas and Vazacopoulos (1998) is applied, which is illustrated in Figure 8. For the selected critical block, it moves one operation in the
middle in the block, highlighted in Figure 8, to the front or end of the block each time, to obtain a new neighbour, representing a new solution. If the obtained new solution has a better objective function, the original solution is updated. When all possible new neighbours have been obtained according to the selected critical block and the solution cannot be optimized, the local search ends.

When finding the critical path, the waiting (operations) times in the buffers are not considered, because waiting times in the buffer are virtual operations and the proposal is to reduce or eliminate them. In classical job-shop scheduling, the critical path always exists, and even in the blocking JSP, the critical path also exists (Zeng et al., 2018). Thus, not accounting for the waiting times in buffers would not influence finding the critical path.

After exchanging the processing sequences in the critical block, the solution may be infeasible. If the newly obtained solution contains a cycle, it will be converted into a feasible solution. The method has been applied successfully in Ren and Wang (2012). Owing to space constraints, it is omitted here, but can be found in Ren and Wang (2012).

### 4.3. Procedure of the proposed TSA

Based on the information above, the proposed algorithm consists of two steps. First, a few feasible solutions are generated quickly, then the obtained solutions are optimized by generating new

---

**Figure 11.** Results obtained by the two-stage algorithm (TSA) and four compared algorithms for scale of instance 15 × 5.

HA = heuristic algorithm; NOIA = novel operations insertionalgorithm; GS = generation scheme; BIH = best insertion heuristic.
neighbours to obtain better solutions. The structure and procedure of the proposed TSA are as follows:

**Step 1.** Parameter settings. Number of initial feasible solutions $k$; build up set $s$ and $s1$; and maximum iteration $GENNO$, $Iter = 1$;

**Step 2.** Obtain $k$ initial feasible solutions using the method proposed in Section 3 and go to Step 3;

**Step 3.** Optimize all solutions one by one. Calculate the critical path of the current solution, clear set $s$, put all operations that need to wait in the buffer in set $s$, sort operations in set $s$ according to the length of the waiting time in descending order, and go to Step 4;

**Step 4.** Select the first operation in set $s$; ensure its next operation (which machine process). If a critical block exists on the same machine as the selected operation, exchange the sequence of operations in the block according to structure N6. Reschedule all operations based on all new possible sequences (based on N6, a critical block can generate more than one neighbourhood) and delete the first operation in set $s$. If a better objective function is obtained, update the original solution, and go back to Step 3; else, if set $s$ is null, go to Step 5; else, restore the sequence in the critical block, and repeat Step 4;

**Step 5.** If all solutions are finished and optimized in the current iteration, add the solution with the best objective function to set $s1$, $Iter = Iter + 1$. If $Iter > GENNO$, go to Step 6; else, if each
Figure 13. Results obtained by the two-stage algorithm (TSA) and four compared algorithms for scale of instance 10 × 10. HA = heuristicalgorithm; NOIA = novel operations insertion algorithm; GS = generationscheme; BIH = best insertion heuristic.

of the two solutions exchange their processing sequences on one or more machines, update
the new solutions, and go back to Step 3;

Step 6. Select the minimum final makespan (best solution) in sl as the optimal makespan of
scheduling. Exit.

5. Experiments and computational results

The experiments conducted to test the algorithms use well-known benchmarks. The benchmarks are
obtained from the standard job-shop benchmark problems La01–40 by Lawrence (1984).

The TSAs are coded in Java and run on a personal computer with an Intel Core™ 2 Quad 2.66 GHz
CPU with 2 GB of RAM.

5.1. Parameter setting

The parameters used in the algorithms are as follows. Parameter $k$, the number of initial feasible
solutions for each instance, is defined as 200; parameter GENNO, the number of iterations to optimize
each initial solution in the local search, is defined as 100; and parameter $p_c$, the probability of two
solutions exchanging their processing sequences on one or more machines, is defined as 0.8. The parameters are set according to the experimental experience in Zeng, Tang, and Yan (2015).

First, LINGO and CPLEX are used to solve the benchmark problem. The instances in which the proportion between the capacity of the buffer and the number of jobs is different are tested under different scenarios. The results obtained by LINGO are listed in Table 2. In the column labelled ‘Size’, 10 × 5 means that there are 10 jobs and that each job has five machine operation stages. ‘Proportion between buffer and jobs’ shows the makespan obtained by LINGO under different proportions between the capacity of the buffer and the number of jobs. For example, if there are 10 jobs and the capacity of each buffer is 2, the proportion is equal to 20%. For the problem on a small scale, which is equal to 10 × 5, LINGO can obtain feasible solutions after 6 h of running. For medium and large scales, however, it could not obtain any feasible solutions. Thus, LINGO may not be suitable to solve the JS-LOB problem. For CPLEX, the computation time is shortened to 2 h, and the results are listed in Supplementary Table S1. By comparison with Supplementary Table S2, it can be seen that CPLEX could not obtain high-quality solutions within the limited computation time, which proves the complexity of the JS-LOB problem.

Next, the benchmark instances were tested using the proposed TSA. The ‘1-Sample Kolmogorov–Smirnov’ non-parametric test in SPSS 18.0 is used to determine whether the selected
Figure 15. Results obtained by the two-stage algorithm (TSA) and four compared algorithms for scale of instance $20 \times 10$. HA = heuristicalgorithm; NOIA = noveloperations insertion algorithm; GS = generation scheme; BIH = bestinsertionheuristic.

benchmark instances conform to a uniform distribution. For example, instance La01 conforms to a uniform distribution in the interval [12, 98]. For each instance, after obtaining an initial feasible solution, optimization continues through a local search. The final results are obtained within 30 s, and are listed in Supplementary Table S2. For example, in first row, ‘1006/890’ represents the initial and final solutions of instance La01 under the proportion $= 0\%$. It was found that, through the local search in the second stage in the proposed algorithm, initial solutions could be improved by 20\%. The situation where the proportion between the buffer and jobs is equal to 100\% was also tested; this is equivalent to the classical JSP. Best Known Solution (BKS) shows the best makespan for the JSP so far. For the small-scale instances, the proposed TSA can obtain a global optimal solution, and the average deviation for all instances is just 1.85\%, proving the effectiveness of the TSA.

5.2. Analysis of the TSA

To further prove the effectiveness of the proposed TSA, it is compared with existing algorithms. Here, four algorithms are selected, namely, the heuristic algorithm (HA), novel operations insertion algorithm (NOIA), generation scheme (GS) and best insertion heuristic (BIH), which are proposed by Brucker et al. (2006), Fahmy, ElMekkawy, and Balakrishnan (2008), Witt and Voß (2010) and Liu et al. (2017), respectively. All the algorithms are implemented under the same conditions. Owing to
limited space, the obtained makespans are listed in the supplementary material. For each instance, the results are obtained within 30 s. The best result obtained by the four algorithms is selected, and compared with the result obtained by the proposed TSA listed in Supplementary Table S2. Then, the percentage deviation between the two results is calculated and shown as gap 1 in Supplementary Table S3. Gap 1 is calculated as follows:

$$\text{Gap 1} = \left( \frac{\text{Best makespan obtained by four algorithms} - \text{Makespan obtained by proposed algorithm}}{\text{Best makespan obtained by four algorithms}} \right) \times 100\%$$

From Supplementary Table S3, it can be seen that the proposed algorithm obtains better results than the four compared algorithms for all benchmark test problems. The average percentage deviations of the makespan obtained by the best of the four algorithms compared with the TSA under different proportions between the buffer and jobs are equal to 0.98%, 1.26%, 5.14%, 5.16%, 5.04% and 4.85%, respectively. These results demonstrate the effectiveness of the proposed TSA, especially when the proportion is larger than 20%.

For further analysis of the speed of convergence and stability, instances from different scales are selected; for example, instance La01 is used to analyse the scale of $10 \times 5$ and La06 to analyse the scale of $15 \times 5$. The scales of instances are divided into three levels: $10 \times 5$, $15 \times 5$ and $20 \times 5$ are...
Figure 17. Results obtained by the two-stage algorithm (TSA) and four compared algorithms for scale of instance 15 × 15. HA = heuristical algorithm; NOIA = novel operations insertion algorithm; GS = generation scheme; BIH = best insertion heuristic.

treated as small level; 10 × 10, 15 × 10 and 20 × 10 are treated as medium level; and 30 × 10 and 15 × 15 are treated as large level. As mentioned already, the results listed in Supplementary Table S2 are obtained within 30 s; for the selected instances at the three different levels, the computational times are extended to 60, 90 and 120 s, respectively, and the results are recorded every 10 s after 30 s until the end of the calculation. The results obtained for proposed TSA and the four compared algorithms are shown in Figures 10–17. In each figure, the curved line ‘TSA’ represents the results obtained by the proposed TSA. ‘P = 0%’ means that the results are obtained under the situation where the proportion between the buffer and jobs is equal to 0. Owing to limited space, the detailed results for each selected instance are listed in the supplementary material.

From Figures 10–17, it can be seen that for all scales of instances, the proposed TSA can generate better solutions than the four compared algorithms, and has a faster convergence rate. For the selected instance in the scale of 15 × 15, the proposed TSA converges between 70 and 80 s, while the four compared algorithms converge between 90 and 100 s. This proves the efficiency and stability of the proposed TSA.

Based on Supplementary Table S2, the influence of the capacity of the buffer is analysed. The average gap 2 of instances for different scales under different scenarios is calculated. The results are listed
Table 3. Results for deviations between the current proportion and 80% proportion.

| Instance Size | Proportion between buffer and jobs (gap 2) |
|---------------|------------------------------------------|
|               | 0%  | 10% | 20% | 30% | 50% |
| La01–05 10×5 | 30.76 | 16.68 | 1.64 | 1.32 | 0.59 |
| La06–10 15×5 | 33.58 | 10.3 | 0.34 | 0.14 | 0.14 |
| La11–15 20×5 | 32.04 | 12.81 | 0.38 | 0.25 | 0.12 |
| La16–20 10×10 | 39.17 | 17.04 | 1.53 | 1.02 | 0.31 |
| La21–25 15×10 | 61.55 | 33.31 | 2.38 | 1.21 | 0.58 |
| La26–30 20×10 | 74.98 | 32.26 | 1.39 | 0.69 | 0.14 |
| La31–35 30×10 | 87.81 | 25.69 | 2.78 | 0.94 | 0.52 |
| La36–40 15×15 | 56.07 | 31.45 | 3.78 | 1.38 | 0.4 |

In Table 3, Gap 2 is calculated as follows:

\[
\text{Gap 2} = \frac{(\text{Makespan under current proportion} - \text{Makespan under proportion 80%})}{\text{Makespan under proportion 80%}} \times 100\%
\]

The results in Table 3 show that the proposed TSA can successfully solve the JS-LOB problem. Comparison with four existing algorithms demonstrates the effectiveness of the proposed TSA, especially when the proportion is larger than 20%. By analysing the influence of the capacity of the buffer, it is found that when the processing time of jobs conforms to a uniform distribution, by increasing the capacity of the buffer, when the proportion between the capacity of the buffer and the number of jobs reaches 20%, the deviation will be less than 4%, after which the influence of increasing the capacity of the buffer will be very small.

6. Conclusions

This article addresses the JS-LOB problem with the objective of minimizing the process makespan. An INLP model is proposed to describe the problem. According to the model, a TSA consisting of obtaining feasible solutions and a local search is proposed to solve the JS-LOB problem. The local search has two operators. The first operator is a neighbourhood structure based on a disjunctive graph model, focused on reducing or eliminating the waiting time of operations in the buffer. The second operator is similar to crossover in the genetic algorithm, to avoid falling into the local optimum. Computational results are presented for a set of benchmark tests, some of which are enlarged by different proportions between the capacity of the buffer and the number of jobs, and the effectiveness of the proposed algorithm is demonstrated by comparing it with four existing algorithms. By analysing the influence of the capacity of the buffer, when the processing time of the job conforms to a uniform distribution and when the proportion between the capacity of the buffer and the number of jobs is larger than 20%, the influence of the buffer will become very small. From the article, if the space in a workshop is limited, 20% could be a good reference point for the proportion of the capacity of the buffer to the number of jobs.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This article is financially supported by NSFC projects [no. 71702028, 71420107028, 71501027], Post Doctoral Science Fund [no. 2017M621154], the 111 Project [B16009], the National Key Research and Development Program of China [2016YFB0901900], NSFC projects [no. 71621061, 71790614, 71520107004] and State Key Laboratory of Synthetical Automation for Process Industries Fundamental Research Funds [grant no. 2013ZCX02].
References

Balas, E., and A. Vazacopoulos. 1998. “Guided Local Search with Shifting Bottleneck for Job Shop Scheduling.” *Management Science* 44 (2): 262–275.

Blazewicz, J., E. Pesch, and M. Sterna. 2000. “The Disjunctive Graph Machine Representation of the Job Shop Scheduling Problem.” *European Journal of Operational Research* 127 (2): 317–331.

Brucker, P., S. Heitmann, J. Hurink, and T. Nieberg. 2006. “Job-Shop Scheduling with Limited Capacity Buffers.” *OR Spectrum* 28 (2): 151–176.

Fahmy, S. A., T. Y. ElMekkawy, and S. Balakrishnan. 2008. “Deadlock-Free Scheduling of Flexible Job Shops with Limited Capacity Buffers.” *European Journal of Industrial Engineering* 2 (3): 231–252.

Lawrence, S. 1984. “Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement).” Ph.D. thesis, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA.

Liu, S. Q., E. Kozan, M. Masoud, Y. Zhang, and F. T. S. Chan. 2018. “Job Shop Scheduling with a Combination of Four Buffering Constraints.” *International Journal of Production Research* 56 (9): 3274–3293.

Mascis, A., and D. Pacciarelli. 2002. “Job-Shop Scheduling with Blocking and No-Wait Constraints.” *European Journal of Operational Research* 143 (3): 498–517.

Nawaz, M., E. E. J. Enscore, and I. Ham. 1983. “A Heuristic Algorithm for the M-Machine, N-Job Flow-Shop Sequencing Problem.” *OMEGA-International Journal of Management Science* 11 (1): 91–95.

Pan, Q. K., L. Wang, and L. Gao. 2011. “A Chaotic Harmony Search Algorithm for the Flow Shop Scheduling Problem with Limited Buffers.” *Applied Soft Computing* 11 (8): 5270–5280.

Pan, Q. K., L. Wang, L. Gao, and W. D. Li. 2011. “An Effective Hybrid Discrete Differential Evolution Algorithm for the Flow Shop Scheduling with Intermediate Buffers.” *Information Sciences* 181 (3): 668–685.

Ren, Q. D. E. J., and Y. P. Wang. 2012. “A New Hybrid Genetic Algorithm for Job Shop Scheduling Problem.” *Computers & Operations Research* 39 (10): 2291–2299.

Sun, J. U. 2009. “A Genetic Algorithm for a Re-entrant Job-Shop Scheduling Problem with Sequence-Dependent Setup Times.” *Engineering Optimization* 41 (6): 505–520.

Tang, J. F., C. K. Zeng, and Z. D. Pan. 2016. “Auction-Based Cooperation Mechanism to Parts Scheduling for Flexible Job Shop with Inter-Cells.” *Applied Soft Computing* 49: 590–602.

Tokola, H., L. Ahlroth, and E. Niemi. 2014. “A Comparison of Rescheduling Policies for Online Flow Shops to Minimize Tardiness.” *Engineering Optimization* 46 (2): 165–180.

Witt, A., and S. Voß. 2010. “Job Shop Scheduling with Buffer Constraints and Jobs Consuming Variable Buffer Space.” *Lecture Notes in Business Information Processing* 46: 295–307.

Zeng, C. K., J. F. Tang, and Z. P. Fan. 2018. “Auction-Based Cooperation Mechanism for Cell Part Scheduling with Transportation Capacity Constraint.” *International Journal of Production Research*. doi:10.1080/00207543.2018.1513660

Zeng, C. K., J. F. Tang, Z. P. Fan, and C. J. Yan. 2018. “Auction-Based Approach for a Flexible Job Shop Scheduling Problem with Multiple Process Plans.” *Engineering Optimization*. doi:10.1080/0305215X.2018.1561884

Zeng, C. K., J. F. Tang, and C. J. Yan. 2014. “Scheduling of No Buffer Job Shop Cells with Blocking Constraints and Automated Guided Vehicles.” *Applied Soft Computing* 24: 1033–1046.

Zeng, C. K., J. F. Tang, and C. J. Yan. 2015. “Job-Shop Cell-Scheduling Problem with Inter-Cell Moves and Automated Guided Vehicles.” *Journal of Intelligent Manufacturing* 26 (5): 845–859.

Zhu, J., X. Li, and Q. Wang. 2009. “Complete Local Search with Limited Memory Algorithm for No-Wait Job Shops to Minimize Make-Span.” *European Journal of Operational Research* 198 (2): 378–386.