A hybrid shifting bottleneck procedure algorithm for the parallel-machine job-shop scheduling problem

Shi Qiang Liu and Erhan Kozan[*]
School of Mathematical Sciences, Queensland University of Technology
2 George St GPO Box 2434, Brisbane Qld 4001 Australia

Abstract: In practice, *parallel-machine job-shop scheduling* (PMJSS) is very useful in the development of standard modelling approaches and generic solution techniques for many real-world scheduling problems. In this paper, based on the analysis of structural properties in an *extended disjunctive graph model*, a *hybrid shifting bottleneck procedure* (HSBP) algorithm combined with Tabu Search metaheuristic algorithm is developed to deal with the PMJSS problem. The original-version SBP algorithm for the *job-shop scheduling* (JSS) has been significantly improved to solve the PMJSS problem with four novelties: *i)* a topological-sequence algorithm is proposed to decompose the PMJSS problem into a set of *single-machine scheduling* (SMS) and/or *parallel-machine scheduling* (PMS) subproblems; *ii)* a *modified Carlier algorithm* based on the proposed lemmas and the proofs is developed to solve the SMS subproblem; *iii)* the *Jackson rule* is extended to solve the PMS subproblem; *iv)* a *Tabu Search metaheuristic algorithm* is embedded under the framework of SBP to optimise the JSS and PMJSS cases. The computational experiments show that the proposed HSBP is very efficient in solving the JSS and PMJSS problems.

Keywords: Scheduling; heuristics; parallel-machine; job-shop; shifting bottleneck procedure; metaheuristics

---

Glossary:

FSS	Flow-Shop Scheduling
JSS	Job-Shop Scheduling
PMJSS	Parallel-Machine Job-Shop Scheduling
SBP	Shifting Bottleneck Procedure
HSBP	Hybrid Shifting Bottleneck Procedure
SMS	Single-Machine Scheduling
PMS	Parallel-Machine Scheduling
DG	Disjunctive Graph
PDG	Partial Disjunctive Graph
DDG	Directed Disjunctive Graph
PDDG	Partial Directed Disjunctive Graph

[*]Corresponding author. Tel: +61 7 3138 1029; Fax: +61 7 3138 2310.
Email addresses: e.kozan@qut.edu.au (E. Kozan).

1 Introduction

In the literature, the *Parallel-Machine Job-Shop Scheduling* (PMJSS) problem has been studied under different names such as the generalised, flexible or complex job shop scheduling problem. Some existing contributions to PMJSS or flexible (complex) job shop scheduling problem are given in the following.

Sadeh et al. (1995) studied a version of the job shop scheduling problem in which some operations have to be scheduled within time windows (i.e. earliest/latest possible start time windows). They developed a new look-back schemes that help the search procedure recover from so-called dead-end search states. Dauzère-Pérès and Paulli (1997) considered an important extension of the classical job-shop scheduling problem, in which the same operation can be performed on more than one machine. They presented an integrated tabu search algorithm by defining a neighborhood structure where there is no distinction between reassigning or resequencing an operation. Chen and Luh (2003) developed a new Lagrangian relaxation approach for the parallel-machine job shop scheduling problem. In the approach, operation precedence constraints rather than machine capacity constraints are relaxed. The relaxed problem is decomposed into single or parallel machine scheduling subproblems. The dual subproblems are solved by using a so-called surrogate subgradient method that allows approximate optimisation of the subproblems. Alvarez-Valdes et al. (2005) described the design and implementation of a scheduling system in a glass factory. The structure basically corresponds to a flexible job-shop scheduling problem with some special characteristics. This system can produce approximate solutions in very short computing times. Xia and Wu (2005) proposed a particle swarm optimization algorithm for the multi-objective flexible job-shop scheduling problem, by combining local search (by self experience) and global search (by neighboring experience). Fattahi et al. (2007) developed a mathematical model and heuristic approaches for flexible job shop scheduling problems. Mathematical model is used to achieve optimal solution for small size problems. Two heuristics approaches are developed to solve the real size problems. Gao et al. (2008) addressed the flexible job shop scheduling problem with three objectives: min makespan, min maximal machine workload and min total workload. They developed a hybrid genetic algorithm (GA) for the problem. This GA algorithm uses two vectors to represent solutions. Advanced crossover and mutation operators are used to adapt to the special chromosome structure and the characteristics of the problem. An extensive computational study on 181 benchmark problems shows the performance of their GA approach.

Some previous research on the SBP algorithm is also summarised as follows. In the literature, Adams *et al.* (1988) initially proposed the SBP algorithm for solving the *job-shop scheduling* (JSS) problem. Because SBP has a very good balance between computational complexity and the quality of generated schedules, it receives considerable interest from other researchers since its introduction. For example, Ramubhin and Marier (1996) generalised the SBP algorithm to solve various types of scheduling problems including open-shop, assembly shops and shops where only a partial ordering on operations pertaining to each job or machine is specified. Ivens and Lambrecht (1996) examined several extensions of the SBP algorithm towards real-life JSS applications by considering some factors such as assembly structure, overlapping operation, setup times and transportation times. Balas and Vazacopoulos (1998) developed a hybrid procedure that embeds a guided local search into a SBP framework to solve the JSS problem. Cheng, Karuno and Kise (2001) developed a SBP approach for the parallel-machine flow-shop scheduling problem. After decomposition, the parallel-machine sub-problem is approximately solved by a constructive heuristic algorithm based on a property of reversibility. Huang and Yin (2004) proposed an improved SBP algorithm for the JSS problem based on proving a theorem of SBP to guarantee solution feasibility. In addition, a refined version SBP that combines this improved SBP with the strategy of back tracking was developed. Mason et al. (2002) developed a modified SBP algorithm for minimizing the total weighted tardiness in a semiconductor wafer fabrication facility. This complex job shop is characterized by re-entrant or re-circulating product flow through a number of different tool groups (one or more machines operating in parallel) which typically contain batching machines, as well as machines that are subject to sequence-dependent setups. The disjunctive graph of the complex job shop is presented along with a description of the proposed heuristic. Their results indicate the heuristic's potential for promoting on-time deliveries by semiconductor manufacturers for their customers' orders. Pfund et al. (2008) also modelled a semiconductor wafer fabrication process as a complex job shop and adapted a modified SBP algorithm to facilitate the multi-criteria optimization of makespan, cycle time, and total weighted tardiness using a desirability function. The desirability function is implemented at two different levels: the subproblem solution procedure level (SSP level) and the machine criticality measure level (MCM level). Monch and Drießel (2005) also considered a modified SBP for complex job shops for semiconductor wafer facilities, which contain parallel batching machines, machines with sequence-dependent setup times and re-entrant process flows. They proposed a two-layer hierarchical approach in order to decompose the overall scheduling problem. The upper layer works on an aggregated model that determines start dates and due dates for the jobs within each single work area, which is defined as a set of parallel machine groups. The lower layer uses the start dates and planned due dates in order to apply shifting bottleneck heuristic type solution approaches for the jobs in each single work area. They conducted simulation experiments in a dynamic job shop environment to assess the performance of the heuristic. Furthermore, Monch et al. (2007) extended the previous research in which only dispatching-based subproblem solution procedures were implemented. Thus in the extended research, they applied more sophisticated near-to-optimal subproblem solution procedures (i.e. genetic algorithms) to solve the parallel-machine scheduling subproblems. Based on simulation experiments, it is indicated that using genetic algorithms leads to improved results in comparison with dispatching-based subproblem solution procedures.

To the best of our knowledge, however, very few researchers addressed the application of the SBP algorithm combined with metaheuristics to solve the PMJSS problem. In this paper, with exploiting the structural properties of the PMJSS problem based on an extended disjunctive graph, we proposed a state-of-the-art hybrid SBP algorithm with several innovative aspects to solve the PMJSS problem efficiently.

This paper is organised as follows. In Section 2, the mathematical programming formulation is given for describing the properties of PMJSS. In Section 3, we proposed an *extended* disjunctive graph model to thoroughly analyse the structural properties of the PMJSS problem. In Section 4, the framework of the proposed hybrid SBP algorithm is described. In Section 5, a topological-sequence algorithm is proposed for decomposing the PMJSS problem into a set of *single-machine scheduling* (SMS) and/or *parallel-machine scheduling* (PMS) subproblems with different release times and delivery times. After decomposing, the SMS and PMS subproblems are mathematically formulated and analysed in Section 6. In Section 7, based on the proposed five lemmas, a modified Carlier algorithm is proposed for quickly solving the SMS subproblem. In Section 8, we extended Jackson's rule to deal with the PMS subproblem. In Section 9, the design of the embedded Tabu Search metaheuristic algorithm under the SBP framework is briefly addressed. We report the computational results in Section 10. Section 11 concludes this paper. The proofs for the proposed lemmas are given in the appendixes.

2 Mathematical Programming Formulation for PMJSS

Notations

n	number of jobs
m	number of machines
J_i	job i ($i = 1, 2, \ldots n$).
M_k	machine k ($k = 1, 2, \ldots, m$).
h_k	number of units of machine k; default is single machine $h_k = 1$.
u_l	the l^{th} unit of machine k ($l = 1, \ldots, h_k$).
o	index of sequence position of operation in one job ($o = 1, 2, \ldots, m$).
e_{ilk}	starting time of job i on the l^{th} unit of machine k.
p_{ilk}	processing time of job i on the l^{th} unit of machine k.
r_{iolk}	= 1, if the o^{th} operation of job i requires the l^{th} unit of machine k; = 0, otherwise.
x_{ilk}	= 1, if job i is assigned to the l^{th} unit of machine k; = 0, otherwise.
y_{ijlk}	= 1, if both jobs i and j are assigned to the l^{th} unit of machine k and job i precedes job j (not necessarily immediately); = 0, otherwise.
$C_{\max}$	maximum completion time or makespan.
L	a very large positive number.

With analysing the characteristics of PMJSS, the mathematical programming formulation for PMJSS is proposed as follows:

PMJSS Mathematical Model

The objective function is to minimise the makespan.

$$\textit{Minimise } C_{\max} \tag{1}$$

Subject to:

$$\sum_{l=1}^{h_k}\sum_{k=1}^{m} r_{iolk}(e_{ilk} + p_{ilk}) \le \sum_{l=1}^{h_k}\sum_{k=1}^{m} r_{i,o+1,l,k}e_{ilk}, o = 1, 2, \ldots, m-1, \forall i. \tag{2}$$

Equation (2) restricts the starting time of (o+1)th operation of job i to be no earlier than its finish time of the o^{th} operation of job i.

$$e_{ilk} \ge e_{jlk} + p_{jlk} + L(y_{ijlk} - 1) \ \forall i, j, l, k. \tag{3}$$

Equation (3) restricts that both jobs i and j are processed on the l^{th} unit of machine k and job i precedes job j (not necessarily immediately).

$$e_{jlk} \ge e_{ilk} + p_{ilk} + L(y_{jilk} - 1) \ \forall i, j, l, k. \tag{4}$$

Equation (4) restricts that that both jobs i and j are processed on the l^{th} unit of machine k and job j precedes job i (not necessarily immediately).

$$y_{ijlk} + y_{jilk} \leq 1 \; \forall i, j, l, k. \tag{5}$$

Equation (5) restricts that conditions that job j precedes job i or job i precedes job j at the l^{th} unit of machine k are exclusive.

$$\sum_{l=1}^{h_k} \sum_{k=1}^{m} x_{ilk} = 1 \text{ and } x_{ilk} + x_{jlk} - 1 \leq y_{ijlk} + y_{jilk} \;\; \forall i, j, l, k. \tag{6}$$

Equation (6) restricts that each machine unit can process at most one job at a time.

$$\sum_{l=1}^{h_k} \sum_{k=1}^{m} r_{imlk}(e_{ilk} + p_{ilk}) \leq C_{\max} \;\; \forall i. \tag{7}$$

Equation (7) restricts that the completion time of the m^{th} (i.e. last) operation of each job is no earlier than the makespan.

$$x_{ilk}, r_{iolk}, \; y_{ijlk} = 0 \text{ or } 1 \;\; \forall i, j, l, k. \text{ and } e_{ilk}, p_{ilk} \geq 0 \;\; \forall i, l, k. \tag{8}$$

Equation (8) satisfies non-negativity and binary-variable conditions.

3 An Extended Disjunctive Graph

For illustrating the extended disjunctive graph model that is proposed for analysing the PMJSS problem, a numerical example is modelled by an extended activity-on-the-node *disjunctive graph* $DG = (\mathrm{O}, C, E)$ shown in Figure 1. The description for this extended *DG* model is given as follows.

Assuming that there are n jobs and m same-type machines in DG, we number the nodes from 1 to N, where $N = n \times \sum_{k=1}^{m} h_k$ denotes the total number of *actual* and *void* operations in DG and h_k is the number of units for machine k. If one job is not processed on a machine, the corresponding operation is defined as "*void*". If one operation is *void*, its processing time is defined as zero. An arc (i, j) connects two *actual* nodes i and j, implying that operation i has to be processed immediately before operation j. Each arc has the length of p_i corresponding to the processing time of operation i. In $DG = (O, C, E)$, O is the set of nodes; C is the set of *conjunctive arcs* representing the fixed precedence relation between each pair of operations belonging to the same job and the directed arcs for connecting the virtual start (end) node with the first (last) operation of each job; E is the set of disjunctive arcs, each of which representing undetermined precedence relation between two operations processed on the same machine. For example, in Figure 1 for job 1 (J_1) that has a prescribed machine sequence from machine M_A to machine M_H, the set of conjunctive arcs for J_1 can be listed as:

$$O_{A1} \rightarrow O_{B1} \rightarrow O_{C1} \rightarrow O_{D1} \rightarrow O_{E1} \rightarrow O_{F1} \rightarrow O_{G1} \rightarrow O_{H1}.$$

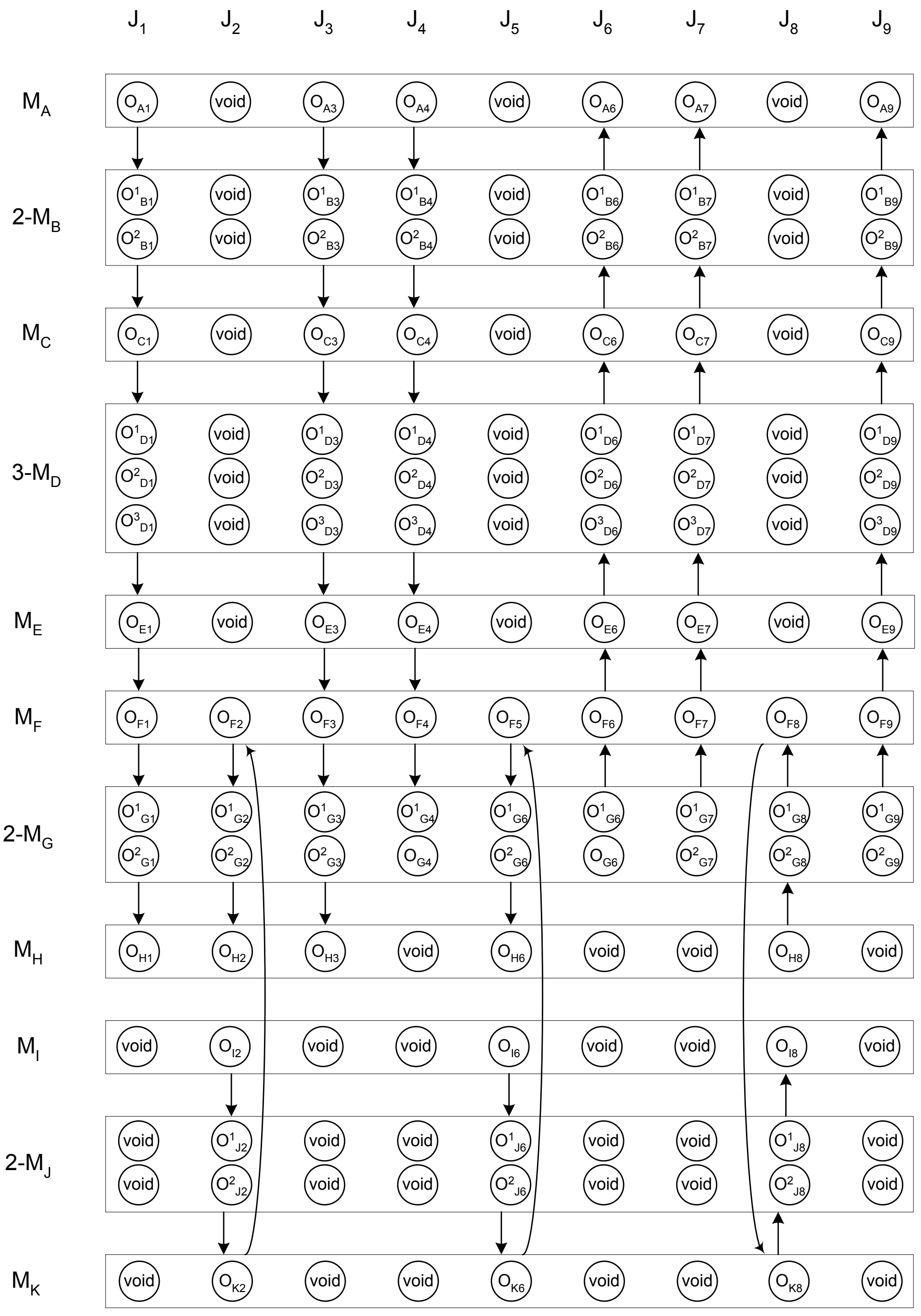

Figure 1 An extended disjunctive graph model proposed for PMJSS, in which only conjunctive arcs are shown in this graph for clarity.

For the sake of figure's clarity, only the conjunctive arcs are drawn in Figure 1. To complete the graph model, for the above PMJSS instance, the rest of the arcs that need to be added into *DG* are separately explained and analysed in the following paragraphs.

Two *virtual* nodes (nodes F and L), which represent the *start* and the *end* of a schedule, needed to added into Figure 1 respectively. The directed arcs for connecting the virtual start (end) node with the first (last) operation of each job also need to be added into *DG*. On the other hand, E denoted as the set of pairs of *disjunctive arcs*, which represent undirected precedence relationships between each pair of two operations processed on the same machine, should also be added into Figure 1.

For each machine in Figure 1, the nodes encircled by a rectangle represent the operations that are processed on the same-type machine. For example, on Machine M_A, the encircled *actual* nodes consist of O_{A1}, O_{A3}, O_{A4}, O_{A6}, O_{A7}, O_{A9}. Note that the set of disjunctive arcs connecting these encircled nodes are omitted in Figure 1 due to its complexity. The number of disjunctive arcs on a single-unit machine is equal to $n'_K(n'_K - 1)$, where n'_K is the number of jobs processed on a singe-unit machine M_K. For example, the subset of disjunctive arcs (E_A) on machine M_A are enumerated in Table 1. In this case, the number of total disjunctive arcs in subset E_A is 6*(6-1)=30.

Table 1 The disjunctive arcs on a single-unit machine with six jobs

Nodes	Disjunctive arcs on a single-unit machine
O_{A1}	$O_{A1} \rightarrow O_{A3}$; $O_{A1} \rightarrow O_{A4}$; $O_{A1} \rightarrow O_{A6}$; $O_{A1} \rightarrow O_{A7}$; $O_{A1} \rightarrow O_{A9}$;
O_{A3}	$O_{A3} \rightarrow O_{A1}$; $O_{A3} \rightarrow O_{A4}$; $O_{A3} \rightarrow O_{A6}$; $O_{A3} \rightarrow O_{A7}$; $O_{A3} \rightarrow O_{A9}$;
O_{A4}	$O_{A4} \rightarrow O_{A1}$; $O_{A4} \rightarrow O_{A3}$; $O_{A4} \rightarrow O_{A6}$; $O_{A4} \rightarrow O_{A7}$; $O_{A4} \rightarrow O_{A9}$;
O_{A6}	$O_{A6} \rightarrow O_{A1}$; $O_{A6} \rightarrow O_{A3}$; $O_{A6} \rightarrow O_{A4}$; $O_{A6} \rightarrow O_{A7}$; $O_{A6} \rightarrow O_{A9}$;
O_{A7}	$O_{A7} \rightarrow O_{A1}$; $O_{A7} \rightarrow O_{A3}$; $O_{A7} \rightarrow O_{A4}$; $O_{A7} \rightarrow O_{A6}$; $O_{A7} \rightarrow O_{A9}$;
O_{A9}	$O_{A9} \rightarrow O_{A1}$; $O_{A9} \rightarrow O_{A3}$; $O_{A9} \rightarrow O_{A4}$; $O_{A9} \rightarrow O_{A6}$; $O_{A9} \rightarrow O_{A7}$;

The characteristics of disjunctive arcs on a multiple-unit machine (i.e. when $h_k > 1$) are much more complicated. According to our analysis, the number of disjunctive arcs on a multiple-unit machine is equal to $n'_K(n'_K - 1)h_k{}^2$. For simplicity, we consider a double-unit ($h_k = 2$) machine M_J on which only three jobs ($n'_K = 3$) are processed. The total disjunctive arcs of subset E_J on a double-unit machine 2-M_J are listed in Table 2, where O^1_{J6} and O^2_{J6} are the same-type operation of job J_6 but processed on two different units of machine 2-M_J. Operations O^1_{J6} and O^2_{J6} are highlighted by superscripts 1 and 2 for indicating Unit 1 and Unit 2 of the double-unit machine 2-M_J. In this case, the number of total disjunctive arcs in subset E_J equals 3*(3-1)*2^2=24.

Table 2 The disjunctive arcs on a double-unit machine with three jobs

Nodes	Disjunctive arcs on a double-unit machine
O^1_{J2}	$O^1_{J2} \rightarrow O^1_{J6}$; $O^1_{J2} \rightarrow O^1_{J8}$; $O^1_{J2} \rightarrow O^2_{J6}$; $O^1_{J2} \rightarrow O^2_{J8}$;
O^2_{J2}	$O^2_{J2} \rightarrow O^1_{J6}$; $O^2_{J2} \rightarrow O^1_{J8}$; $O^2_{J2} \rightarrow O^2_{J6}$; $O^2_{J2} \rightarrow O^2_{J8}$;
O^1_{J6}	$O^1_{J6} \rightarrow O^1_{J2}$; $O^1_{J6} \rightarrow O^1_{J8}$; $O^1_{J6} \rightarrow O^2_{J2}$; $O^1_{J6} \rightarrow O^2_{J8}$;
O^2_{J6}	$O^2_{J6} \rightarrow O^1_{J2}$; $O^2_{J6} \rightarrow O^1_{J8}$; $O^2_{J6} \rightarrow O^2_{J2}$; $O^2_{J6} \rightarrow O^2_{J8}$;
O^1_{J8}	$O^1_{J8} \rightarrow O^1_{J2}$; $O^1_{J8} \rightarrow O^1_{J6}$; $O^1_{J8} \rightarrow O^2_{J2}$; $O^1_{J8} \rightarrow O^2_{J6}$;
O^2_{J8}	$O^2_{J8} \rightarrow O^1_{J2}$; $O^2_{J8} \rightarrow O^1_{J6}$; $O^2_{J8} \rightarrow O^2_{J2}$; $O^2_{J8} \rightarrow O^2_{J6}$;

With the above analysis, the set of disjunctive arcs (E) can be partitioned into m subsets $E_1, E_2, \ldots\ldots, E_m$, where m is the number of same-type machines and E_k $(\forall k = 1,2,...,m)$ denotes the subset of disjunctive arcs connecting the encircled actual nodes on a machine. Actually, determining a PMJSS schedule corresponds with selecting one direction for each pair of disjunctive arcs and discarding the redundant arcs in E.

Let $E' = \bigcup E'_k (\forall k = 1, 2, \ldots, m)$ be such a compound set of *directed* disjunctive arcs, where E'_k is the subset of directed disjunctive arcs on a machine. As a result, $DDG = (O, C, E')$ denotes the *directed disjunctive graph* representing one schedule. A feasible schedule requires that DDG is acyclic and satisfies all the required constraints. The length of a longest path from the virtual start node to the virtual end node in DDG is equivalent to the makespan of the schedule. .

4 A Hybrid Shifting Bottleneck Procedure Algorithm

Based on the above analysis in this extended disjunctive graph model, a *hybrid shifting bottleneck procedure* (HSBP) algorithm is developed to solve the PMJSS problem.

Firstly, it is essential to build up a *partial disjunctive graph* $PDG = (O, C, \bigcup E'_k \mid k \in M_0, \bigcup E_v \mid v \in M - M_0)$. In *PDG*, M_0 is the subset of machines that have been sequenced; $\bigcup E'_k \mid k \in M_0$ is such a subset of *directed* disjunctive arcs on sequenced machines; and $\bigcup E_v \mid v \in M - M_0$ is the subset of disjunctive arcs on unsequenced machines.

With a given PDG, the PMJSS problem can be decomposed as a set of *single-machine scheduling* (SMS) and/or *parallel-machine scheduling* (PMS) subproblems with different release times and delivery times. After decomposing, the number of the generated SMS and PMS subproblems is respectively equal to the number of unsequenced single-unit machine and unsequenced multiple-unit machines in the PDG.

In this study, the original-version SBP algorithm for the classical JSS problem has been greatly improved to efficiently solve the PMJSS problem in four novelties:

- the topological-sequence algorithm (See Section 5) is proposed to solve the *partial directed disjunctive graph* ($PDDG$) of the PMJSS problem and to decompose the PMJSS problem into a set of SMS and/or PMS subproblems with different release times and delivery times (See Section 6).
- a modified Carlier algorithm (See Section 7) based on the proposed lemmas is developed for solving the SMS subproblems;
- an extended Jackson algorithm (See Section 8) is implemented to solve the PMS subproblems;
- a metaheuristic algorithm (See Section 9) is embedded under the architecture of HSBP to re-optimise the partial sequence and to further optimise the complete sequence after obtaining the complete sequence.

The procedure of the proposed HSBP algorithm for the PMJSS problem is described as follows.

A Hybrid Shifting Bottleneck Procedure Algorithm

```
Step 1:  Set $M_0 = \phi$. The initial partial directed disjunctive graph ($PDDG$)
         contains all the conjunctive arcs and no disjunctive arcs, i.e.
         $PDDG = (O, C)$.
Step 2:  Do the following for each unsequenced machine $v \in M - M_0$:
         Step 2.1:  According to the given PDDG, implement the topological-
                    sequence algorithm (See Section 5 for detail) to decompose the
                    PMJSS problem into a set of SMS and/or PMS subproblems.
```

Step 2.2: Solve the SMS subproblems by the modified Carlier algorithm (See Section 7 for detail).

Step 2.3: Solve the PMS subproblems by the extended Jackson algorithm (See Section 8 for detail).

Step 3: Bottleneck Selection and Sequencing:

Step 3.1: Choose machine k as bottleneck machine by $L_{\max}(k) = \max_{i \in M - M_0} (L_{\max}(i))$.

Step 3.2: Set the job-sequence on machine k according to the results obtained in *Step 2.2* or *2.3*.

Step 3.3: Update $M_0 = M_0 \bigcup \{k\}$.

Step 3.4: Update the *PDDG* by adding E'_k, i.e. $PDDG = (O, C, \bigcup E'_k \mid k \in M_0)$.

Step 3.5: If $M_0 \equiv M$, go to *Step 5*.

Step 4: Re-sequence and re-optimise the current *PDDG* by the metaheuristic algorithm (see Section 9 for detail); then go to *Step 2*.

Step 5: Optimise the complete directed disjunctive graph:

Step 5.1: Optimise the $DDG = (O, C, \bigcup E'_k \mid k \in M)$ by the metaheuristic algorithm (see Section 8 for detail).

Step 5.2: If the solution is equal to the lower bound or the known optimum, then stop; otherwise, run till the stopping conditions of the metaheuristic algorithm are satisfied.

In a word, the HSBP algorithm schedules machines consecutively one at a time. At each HSBP iteration, two key decisions have to be made in *Step 2*, namely, how to decompose the PMJSS problem into a set of SMS and/or PMS subproblems as well as how to solve these subproblems so that the bottleneck machine and its corresponding job sequence can be effectively and efficiently determined.

5 Topological-Sequence Algorithm for Decomposing PMJSS

At each HSBP iteration, we need to build up a partial directed disjunctive graph model, $PDDG = (O, C, \bigcup E'_k \mid k \in M_0)$, which consist of the conjunctive arcs (i.e. C) of operations belonging to the same job and the directed disjunctive arcs (i.e. $\bigcup E'_k$) of operations processed on sequenced machines ($\forall k \in M_0$).

Implemented for decomposing the PMJSS problem, the topological-sequence algorithm was initially proposed by Liu and Ong (2004) to efficiently compute the values of heads e_i (starting times or the longest distance from the virtual start node to this node) and tails l_i (the longest distance from this node to the virtual end node) for each node i ($\forall i \in O$) in a directed disjunctive graph model. The procedure of this algorithm is briefly presented in the following.

Topological-Sequence Algorithm

Step 1: Compute the in-count value (the number of predecessors) of each node.

Step 2: Find a topological sequence as follows:

Step 2.1: Select the virtual start node as the first node on the topological order list.

Step 2.2: Decrement the in-count value for each of the immediate successor nodes of the selected node by 1.

Step 2.3: Select any of the unselected nodes having a zero in-count value and put this node as the next node on the topological order list.

Step 2.4: Repeat *Steps* 2.2 and 2.3 until all nodes are selected.

Step 3: Start from initialising the starting time of the virtual start node and calculate the starting times of all nodes in the topological sequence by

$$e_i = \max\{e_{PM[i]} + p_{PM[i]}, e_{PJ[i]} + p_{PJ[i]}\} \tag{9}$$

where $PM[i]$ is the same-machine operation processed just before operation i, if it exists; $PJ[i]$ is the same-job operation that just precedes operation i, if it exists.

Step 4: Start from initialising the delivery time of the virtual end node and calculate the tail times of all the nodes in the reverse order of the topological sequence by

$$l_i = \max\{l_{SM[i]} + p_{SM[i]}, l_{SJ[i]} + p_{SJ[i]}\} \tag{10}$$

where $SM[i]$ is the same-machine operation processed just after operation i, if it exists; $SJ[i]$ is the same-job operation that immediately follows operation i, if it exists.

6 SMS and PMS Subproblems

After decomposing the PMJSS problems into a set of *single-machine scheduling* (SMS) and/or *parallel-machine scheduling* (PMS) subproblems, the critical issue of *shifting bottleneck procedure* turns to solve the SMS and/or PMS subproblems with the different release times and delivery times. The objective of the SMS and PMS subproblems is to minimise the maximum lateness.

The formulation of the SMS subproblem is described as below. Given a set of jobs $J = \{1, 2, ..., n\}$, each job j ($\forall j \in J$) has release time r_j, processing time p_j and delivery time q_j.

For a given schedule, the lateness L_j of job j is defined as $L_j = C_j - d_j$, that is, the (positive or negative) time difference between the completion time C_j and the due date d_j. The objective function is to minimise the maximum lateness $L_{\max} = \max_{j \in J} L_j = \max_{j \in J}(C_j - d_j)$. By our analysis, for the SMS subproblem decomposed from the PMJSS problem, minimising the maximum lateness is equivalent to minimising the makespan. The proof is given as follows.

As each job j in a SMS subproblem has a non-negative delivery time q_j (the longest distance from node j to the virtual last node in the PMJSS problem), the completion-delivery time is defined as $f_j = C_j + q_j$. Thus, for the SMS subproblem with the different release times and delivery times, the makespan actually equals the maximum completion-delivery time. The equivalence between two objectives can easily be demonstrated by subtracting a sufficiently large constant, that is,

$const = \hat{C}_{\max}$ ($\hat{C}_{\max}$ is the makespan of the *partial directed disjunctive graph* of the PMJSS problem)

$d_j = \hat{C}_{\max} - q_j = const - q_j$

$L_j = C_j - d_j = C_j - (const - q_j) = C_j + q_j - const = f_j - const$

$L_{\max} = \max_{j \in J} L_j = \max_{j \in J}(f_j - const)$

$\min L_{\max} \cong \min f_{\max}$

Therefore, a SMS subproblem decomposed from the PMJSS problem can be mathematically formulated as below. Assume that the current set of sequenced machines is M_0 and one unsequenced machine relevant to this SMS subproblem is machine k ($\forall k \in M - M_0$).

SMS Model: $SMS(k, M_0)$

The objective function is to minimise the makespan ($C^k_{\max}$) of the SMS subproblem.

$$Minimise\ \ C^k_{\max}\ (\forall k \in M - M_0) \tag{11}$$

Subject to:

$$C^k_{\max} \geq e_i + p_i + q_i,\ \forall i \in O_k \tag{12}$$

Equation (12) restricts that the completion times of the operations (i.e. $O_k \subset O$) processed on machine k are no earlier than the makespan of the SMS subproblem.

$$e_i \geq e_j + p_j \vee e_j \geq e_i + p_i,\ \forall i, j \in O_k \tag{13}$$

Equation (13) defines the precedence relationships of jobs i and j ($i \neq j$) processed on machine k.

$$e_i \geq r_i,\ \forall i \in O_k \tag{14}$$

Equation (14) satisfies the job-ready conditions.

Similarly, one PMS subproblem decomposed from the PMJSS problem can be mathematically modelled as below.

PMS Model: $PMS(k, M_0)$

The objective function is to minimise the makespan of the PMS subproblem.

$$Minimise\ \ C^k_{\max}\ (\forall k \in M - M_0) \tag{15}$$

Subject to:

$$C^k_{\max} \geq e_i + p_i + q_i,\ \forall i \in O_k \tag{16}$$

$$e_j \geq e_i + p_i + L(y_{ijz} - 1), \forall i, j \in O_k \tag{17}$$

$$e_i \geq e_j + p_j + L(y_{jiz} - 1), \forall i, j \in O_k \tag{18}$$

$$y_{ijz} + y_{jiz} \leq 1, \forall i, j \in O_k \tag{19}$$

$$x_{iz} + x_{jz} - 1 \leq y_{ijz} + y_{jiz}, \forall i, j \in O_k \tag{20}$$

$$\sum\nolimits_z x_{iz} = 1,\ \forall i \in O_k \tag{21}$$

$$y_{ijz}, y_{jiz}, x_{iz}, x_{jz} = \begin{cases} 0 \\ 1 \end{cases}, \forall i, j \in O_k \tag{22}$$

$$e_i \geq r_i, \forall i \in O_k \tag{23}$$

where x_{iz}= 1 if operation i is assigned to the z^{th} unit of a parallel-machine k, otherwise x_{iz}= 0; y_{ijz} = 1 if both operation i and j are assigned to the z^{th} unit of the parallel-machine k and operation i precedes operation j, y_{ijz} = 0 otherwise; and L is a sufficiently large number. In PMS model, equations (16-23) particularity restricts that the precedence relationship between a pair of operations processed on a unit of the parallel machine is exclusive and each unit can process at most one job at a time.

7 A Modified Carlier Algorithm for SMS

In scheduling theory, for the *basic* single-machine scheduling (SMS) problem in which all the release times and delivery times are set as zero, the objective of minimising the maximum lateness can be achieved by the *earliest due date* (EDD) dispatching rule (Kellerer 2005). However, in the existence of different release times and delivery times, this SMS problem becomes complex.

In the literature, one of efficient approximation algorithms for solving the SMS problem with different release times and delivery times is called *Schrage algorithm* (Carlier 1982). The procedure of the Schrage algorithm is presented as follows.

Schrage Algorithm

Step 1: Initialisation. Set a time point, $t = \min_{i \in I} r_i$. Set $U = \phi$, and $\bar{U} = J$, where U is the set of scheduled jobs; $\bar{U} = J - U$ is the set of unscheduled jobs;

Step 2: At time point t, if $r_j \le t$ and $q_j = \max_{i \in \bar{U}} q_i$ ($\forall j \in \bar{U}$), select job j.

Step 3: Set the starting time of job j, $e_j = t$.

Step 4: Update $U = U \bigcup \{j\}$, $\bar{U} = \bar{U} - \{j\}$, and $t = \max(t_j + p_j, \min_{i \in \bar{U}} r_i)$.

Step 5: If $\bar{U}$ is empty, stop; otherwise, go to *Step 2*;

The SMS schedule obtained by the Schrage algorithm is called a *Schrage schedule*. The properties of the Schrage schedule are summarised below:

- The sequence of jobs $a, a+1, ..., c$ forms a block in a Schrage schedule and is called the *critical sequence* $\Lambda = \{a, a+1, ..., c\}$, if $C_{\max} = e_i + p_i + q_i, \forall i \in \Lambda$.
- Job c is called the *critical job* because it is the last job in the critical sequence Λ.
- Job a is called the *idled job*, i.e. the first job in critical sequence Λ. This is because there may be idle time before a and there must be no idle time between the processing of any pair of jobs in the critical sequence Λ.
- It is obvious for $r_i \ge r_a$, $\forall i \in \Lambda$.
- The job satisfying the property $q_w < q_c$ and having the largest subscript in the critical sequence is called the *interference job* w.

Based on the above analysis, Carlier (1982) proposed two theorems for the Schrage schedule.

Theorem 1 Let $C_{Schrage}$ be the makespan of the Schrage schedule. If the Schrage schedule is not optimal, there is a interference job w and a critical subset $\Lambda_w = \{w+1,...,c\}$ out of critical sequence $\Lambda = \{a, a+1,...,c\}$ such that,

$$h(\Lambda_w) = \min_{j\in\Lambda_w} r_j + \sum_{j\in\Lambda_w} p_j + \min_{j\in\Lambda_w} q_j > C_{Schrage} - p_w \tag{24}$$

Thus, the distance from the makespan of Schrage schedule to the optimal makespan is less than p_w. Moreover, in an optimal schedule, job w will be processed either before or after all the jobs of critical subset $\Lambda_w = \{w+1,...,c\}$. In case 1, interference job w will be processed before Λ_w, by resetting the delivery time of job w as

$$q_w = \max\{q_w, \sum_{j\in\Lambda_w} p_j + q_c\} \tag{25}$$

In case 2, interference job w will be processed after Λ_w by resetting the release time of job w as

$$r_w = \max\{r_w, \min_{j\in\Lambda_w} r_j + \sum_{j\in\Lambda_w} p_j\} \tag{25}$$

Theorem 2 If this Schrage schedule is optimal, there exists Λ_w such that $h(\Lambda_w) = C_{Schrage}$.

However, Theorem 2 proposed by Carlier (1982) may be incorrect, which can be simply proved by an enumerative example given in Appendix 1.

Based on our thorough analysis for the Schrage schedule, five lemmas are proposed below. The proofs for the proposed lemmas are given in Appendixes 2, 3, and 4.

Lemma I For a Schrage schedule with a critical sequence $\Lambda = \{a, a+1,...,c\}$, there exists $C_{\max} = LB(\Lambda) + q_c - \min_{j\in\Lambda} q_j$.

Lemma II If $q_c = \min_{j\in\Lambda} q_j$, then $C_{\max} = LB(\Lambda)$.

Lemma III If there is an interference job w, then $C_{\max} - LB(\Lambda_w) < p_w$.

Lemma IV For a Schrage schedule, there exists $C_{\max} = \min_{j\in\Lambda} e_j + \sum_{j=a}^{c} p_j + \min_{j\in\Lambda} q_j$.

Lemma V Even if $q_c = \min_{j\in\Lambda} q_j$, this Schrage schedule may not be optimal.

In terms of our proposed lemmas, a modified-version Carlier algorithm is developed for efficiently solving the SMS subproblem with different release times and delivery times.

A Modified Carlier Algorithm

The procedure of the proposed modified Carlier algorithm is described as follows:

Step 1: Apply Schrage algorithm to the current SMS instance and then save the results of the obtained Schrage schedule.

Step 2: If there exist the interference job w satisfying $q_w < q_c$, the interference job w will be processed after critical subset Λ_w by setting $r_w = \max\{r_w, \min_{j\in\Lambda_w} r_j + \sum_{j\in\Lambda_w} p_j\}$ and then go to Step 1.

Step 3: Otherwise, among the saved results, choose the best Schrage schedule with the minimum makespan value.

8 An Extended Jackson Algorithm for PMS

In the literature, the *basic* parallel-machine-scheduling (PMS) problem with minimising the maximum tardiness was efficiently solved by a constructive heuristic based on Jackson's rule (Kellerer 2005). Here, we extend Jackson's rule to quickly solve the PMS subproblems with release times and delivery times and with the objective of minimising the makespan. The procedure of an extended Jackson algorithm for quickly solving the PMS subproblem with release times and delivery times is described as follows.

An Extended Jackson Algorithm

```
Step 1: Initialisation. Set the available times of all machine units as zero. Set $U = \phi$,
        and $\bar{U} = J$, where $U$ is the set of scheduled jobs; $\bar{U} = J - U$ is the set of
        unscheduled jobs.
Step 2: While $\bar{U} \neq \phi$:
    Step 2.1: Choose the $l^{th}$ parallel-machine unit that leads to the minimum
              available time $a_{\min}$; break the tie arbitrarily.
    Step 2.2: Initialise a subset of candidate jobs $U_c = \phi$.
    Step 2.3: For each job $j \in \bar{U}$, if $r_j \geq a_{\min}$, then update $U_c = U_c \bigcup \{j\}$.
    Step 2.4: If $U_c \neq \phi$, select the best candidate job by $j^* = \arg\min_{j \in U_c}(r_j)$; and
              then set the starting time of job $j^*$: $e_{j^*} = r_j$.
    Step 2.5: If $U_c \equiv \phi$, select the best candidate job by $j^* = \arg\max_{j \in \bar{U}}(q_j)$;
              and then set the starting time of job $j^*$: $e_{j^*} = a_{\min}$.
    Step 2.6: Update $U = U \bigcup \{j^*\}$ and $\bar{U} = \bar{U} - \{j^*\}$; and update the
              available time of the $l^{th}$ parallel-machine unit:
```

9 Embedded Metaheuristic Algorithm for Re-Sequencing and Re-Optimising

The approaches of neighbourhood search guided by metaheuristics such as *Tabu Search* (TS) offer a comparatively simple implementation but are very effective and efficient to solve the large-size problems with short computational time and good accuracy.

The design of an efficient TS algorithm is a sophisticated art because TS consists of several elements called the *initial solution*, *move*, *neighbourhood*, *searching strategy*, *memory*, *aspiration function*, *stopping rules*, etc. The *move* is a function which transforms a solution into another solution. The subset of moves applicable to a given solution generates a collection of feasible solutions called the *neighbourhood*. At each step, the neighbourhood of the current solution is searched in order to find an appropriate neighbour, typically the best in the neighbourhood. Next, the move that leads to this neighbour is performed and the resulting solution becomes the new current solution to initiate the next step. To avoid cycling or implicitly becoming trapped in a local optimum, a short term memory called a *tabu list* determines moves that are forbidden. Attributes of moves are identified to be recorded on this list, for a chosen span of time, as a way to prevent future moves that would "undo" the effects of previous moves. Nevertheless, a forbidden move may be accepted if an aspiration function evaluates it as sufficiently profitable. The stopping rule that ends the search traditionally consists of setting a limit on the execution time, number of iterations, and number of consecutive iterations without improving the makespan.

The brief description of the TS procedure is given below.

```
Step 1:  Generate an initial solution $\zeta$ .
Step 2:  Initialise the tabu list.
Step 3:  Perform a certain number of TS iterations depending on the problem size:
    Step 3.1:  Build up the neighbourhood based on the current solution $\zeta$ .
    Step 3.2:  Choose the best neighbour $\xi^*$, which is not a tabu or satisfies
               the aspiration criterion.
    Step 3.3:  Set $\zeta = \xi^*$ and update the tabu list. If the stopping condition is
               met, go to Step 4.
Step 4:  Return the best solution found.
```

Hence, to exploit the merits of metaheuristics, it is decided that SBP and TS are combined as a hybrid algorithm for solving the large-size PMJSS problem. Under the architecture of the hybrid algorithm, the metaheuristic algorithm is embedded for re-optimising the partial (and complete) PMJSS schedule at each HSBP iteration. For more details of the metaheuristic methodology, please refer to the papers (Liu and Ong 2004; Liu *et al.* 2005).

10 Computational Experiments

The proposed HSBP algorithm has been coded in Visual C++ and tested on a HP desktop with Pentium IV 3.20 GHz Processor and 2 GB RAM. The computational experiment is to evaluate the effectiveness of the proposed HSBP algorithm and examine the improvement in solution quality when the TS algorithm is embedded under the framework of SBP.

First, the proposed HSBP algorithm without embedding the Tabu Search algorithm is tested on the benchmark JSS data collected by OR-Library (http://people.brunel.ac.uk/~mastjjb/jeb/orlib/). The computational results for these benchmark JSS instances (LA01-40) are summarised in Table 3, in which the deviation of one instance is calculated as the percentage of the obtained makespan away from the optimal value (*Opt*), namely, *Deviation* = 100×(*Makespan* – *Opt*)/*Opt*. The optimal value (*Opt*) is equal to the lower bound (*LB*) value or the best upper bound (*UB*) value known in the literature.

As shown in Table 3, the optimal solutions (most of them equal to the *LB*) of the instances LA06-10, LA11-15, and LA31-35 can be easily found by the proposed HSBP algorithm because the number of jobs is several times larger than the number of machines (i.e. $n/m > 2$). For the other tough instances, the deviation becomes larger and larger when the problem size of the JSS instance increases, implying that the proposed HSBP algorithm without embedding combining the TS algorithm seems not able to meet the demand for high solution quality.

Table 3 Computational results of JSS by the proposed SBP algorithm without TS

JSS Instances	n	m	Average Deviation (%)	Average Time (s)
LA01-05	10	5	2.08	0.156
LA06-10	15	5	0.39	0.281
LA11-15	20	5	0.00	0.453
LA16-20	10	10	10.13	0.829
LA21-25	15	10	7.98	1.672
LA26-30	20	10	6.03	2.734
LA31-35	30	10	0.70	5.746
LA36-40	15	15	15.48	4.961

To verify the superiority of the proposed HSBP algorithm combined with the TS algorithm, the benchmark instances (i.e. LA01-40) are further computed after activating the embedded components of the TS algorithm. Table 4 shows the comparison of the computational outcomes between the SBP algorithm without TS and the HSBP algorithm with TS.

Table 4 Comparison of the computational results of JSS between SBP without TS and HSBP with TS

JSS Instances	n	m	Average Deviation (%)		Average Time (s)	
			SBP	HSBP	SBP	HSBP
LA01-05	10	5	2.08	0.00	0.156	3.716
LA06-10	15	5	0.39	0.00	0.281	13.788
LA11-15	20	5	0.00	0.00	0.453	31.286
LA16-20	10	10	10.13	0.19	0.829	116.146
LA21-25	15	10	7.98	0.36	1.672	328.125
LA26-30	20	10	6.03	0.02	2.734	687.682
LA31-35	30	10	0.70	0.00	5.746	184.807
LA36-40	15	15	15.48	0.84	4.961	829.268

The comparison indicates that the HSBP with TS performs better than the pure SBP, especially for the large-size tough JSS instances. On the other hand, the HSBP algorithm with TS has to spend more running time due to the fact that more computing efforts are made to guarantee to find the better or optimal solutions. With the above comparative study shown in Table 4, it is enough to validate the efficiency of the proposed HSBP algorithm.

Furthermore, the proposed HSBP problem is applied to solve the PMJSS benchmark instances (named PMJSS-LA instances) established by adjusting the benchmark JSS-LA instances. In the PMJSS-LA benchmark instances, the number of units of each machine for 5-machine PMJSS instances (i.e. PMJSS-LA01 to PMJSS-LA15 instances) is $h_k = \{1,2,1,2,1\} \mid k = 1,2,3,4,5$; the number of units of each machine for 10-machine PMJSS instances (i.e. PMJSS-LA16 to PMJSS-LA35 instances) is $h_k = \{1,2,1,2,1,2,1,2,1,2\} \mid k = 1,...,10$; the number of units of each machine for 15-machine PMJSS instances (i.e. PMJSS-LA36 to PMJSS-LA40 instances) is $h_k = \{1,2,1,2,1,2,1,2,1,2,1,2,1,2,1\} \mid k = 1,...,15$.

The computational experiments on PMJSS-LA instances are summarised in Table 5. In Table 5, a collection of PMJSS benchmark instances are created in the first column. In Table 5, the names of forty Lawrence's benchmark JSS instances (i.e. LA01-40) are indicated in the first column. In the second column, *n* and *m* are the number of jobs and machines for each benchmark instance.

Since metaheuristic algorithms cannot guarantee to solve the scheduling problems optimally, the quality of the best solution found can be evaluated by comparing the confirmed optimum, the best upper bound that is the solution provided in the literature but has not been verified as optimum, or the lower bound. The lower bound is used for evaluate the optimality performance of the proposed methodology for PMJSS because there are no benchmark PMJSS results available in the literature.

In the literature, the following equation is well known to calculate the lower bound of the classical JSS problem:

$$LB_{JSS} = \max\{\max_j(\sum_{i=1}^{n} p_{ij}), \max_i(\sum_{j=1}^{m} p_{ij})\}$$

$$(26) LB_{JSS} = \max\{\max_j(\sum_{i=1}^{n} p_{ij}), \max_i(\sum_{j=1}^{m} p_{ij})\}$$

which is the maximum value between the maximum sum of the processing times of all operations of a job and the maximum sum of the processing times of all operations on each machine. Here, p_{ij} is the processing time of an operation of Job i on Machine j M_j.

Here, we propose a new formula to calculate the lower bound of the PMJSS problem. This lower bound calculated is applied as one of stopping conditions in our proposed HSBP algorithm.

$$LB_{PMJSS} = \max\{\max_j(\sum_{i=1}^{n} p_{ij}), \max_i(\sum_{j=1}^{m} p_{ij} \mid h_k = 1)\} \qquad (27)$$

$LB_{CBPMJSS} = \max\{\max_j(\sum_{i=1}^{n} p_{ij}), \max_i(\sum_{j=1}^{m} p_{ij} | if\ u_j = 1)\}$ The lower bound of PMJSS is the maximum value between the maximum sum of the processing times of all operations of a job and the maximum sum of the processing times of all operations on a single-unit machine.

Thus, the third column (*LB_{JSS}*) presents the lower bounds calculated by Eq. (26). The forth column (*Opt_{JSS}*) gives the optimal solution of the JSS-LA instances.

The fifth column (*LB_{PMJSS}*) gives the lower bound instances calculated by Eq. (27) for PMJSS. Obviously, the lower bound values of PMJSS instances are usually smaller than (or equal to) those of JSS instances. The sixth column (*Solution*) provides the best solutions of the LA-PMJSS instances obtained by our proposed HSBP algorithm. In the seventh column, the CPU times of the metaheuristic algorithm are reported with the measurement unit of second. The eighth column (*Gap1*) shows the relative gap between our obtained PMJSS results and the lower bounds of PMJSS-LA instances, i.e. ($Solution-LB_{PMJSS})/LB_{PMJSS}\times 100$. The average deviation from the lower bound value is 8.17%, implying that the proposed methodology can find the solutions that are very close to optimum. The ninth column (Improve) shows the relative gap between our obtained PMJSS results and the optimal solutions of JSS-LA instances, i.e. $(Opt_{JSS}-Solution)/Opt_{JSS}\times 100$. This comparison is very meaningful because it implies that the efficiency can be improved by at least 2.07% on average in a more realistic manufacturing or logistic system in which some operations can be performed on more than one machine.

Table 5 Computational results of PMJSS based on the PMJSS-LA benchmark instances

Instances	Size	JSS-LA		PMJSS-LA				
		LB_{JSS}	Opt_{JSS}	LB_{PMJSS}	Solution	Time (s)	Deviation (%)	Improve (%)
LA01	10*5	666	666	666	666	63.897	0.00	0.00
LA02	10*5	635	655	597	604	39.235	1.17	7.79
LA03	10*5	588	597	515	588	43.719	14.17	1.51
LA04	10*5	537	590	537	590	36.993	9.87	0.00
LA05	10*5	593	593	593	593	40.356	0.00	0.00
LA06	15*5	926	926	926	926	184.965	0.00	0.00
LA07	15*5	869	890	869	887	190.578	2.07	0.34
LA08	15*5	863	863	863	863	191.691	0.00	0.00
LA09	15*5	951	951	890	890	188.328	0.00	6.41
LA10	15*5	958	958	881	895	227.563	1.59	6.58
LA11	20*5	1222	1222	1222	1222	628.881	0.00	0.00
LA12	20*5	1039	1039	1027	1030	604.219	0.29	0.87
LA13	20*5	1150	1150	1150	1150	594.136	0.00	0.00
LA14	20*5	1292	1292	1132	1155	649.059	2.03	10.60
LA15	20*5	1207	1207	1207	1207	605.785	0.00	0.00
LA16	10*10	717	945	717	904	131.157	26.08	4.34
LA17	10*10	683	784	646	745	104.253	15.33	4.97
LA18	10*10	663	848	663	808	131.157	21.87	4.72
LA19	10*10	685	842	685	818	104.253	19.42	2.85
LA20	10*10	756	902	756	872	104.253	15.34	3.33
LA21	15*10	1040	1048	935	1025	524.628	9.63	2.19
LA22	15*10	830	927	800	927	497.724	15.88	0.00
LA23	15*10	1032	1032	1032	1032	424.628	0.00	0.00
LA24	15*10	857	935	846	928	524.628	9.69	0.75
LA25	15*10	864	977	792	974	628.881	22.98	0.31
LA26	20*10	1218	1218	1218	1218	1730.824	0.00	0.00
LA27	20*10	1235	1242	1188	1212	1972.961	2.02	2.42
LA28	20*10	1216	1216	1142	1208	1678.137	5.78	0.66
LA29	20*10	1120	1182	1017	1117	1809.294	9.83	5.50
LA30	20*10	1355	1355	1355	1355	1705.041	0.00	0.00
LA31	30*10	1784	1784	1784	1784	1466.913	0.00	0.00
LA32	30*10	1850	1850	1850	1850	1233.677	0.00	0.00
LA33	30*10	1719	1719	1585	1670	3469.087	5.36	2.85
LA34	30*10	1721	1721	1721	1721	1870.473	0.00	0.00
LA35	30*10	1888	1888	1888	1888	1696.025	0.00	0.00
LA36	15*15	1028	1268	1028	1213	6022.352	18.00	4.34
LA37	15*15	986	1397	986	1366	6095.448	38.54	2.22
LA38	15*15	1171	1203	943	1195	6075.039	26.72	0.67
LA39	15*15	1012	1233	1012	1176	6049.256	16.21	4.62
LA40	15*15	1222	1228	1027	1202	6028.847	17.04	2.12
Average							8.17	2.07

11 Conclusions

In this paper, we investigate the *parallel-machine job-shop scheduling* (PMJSS) problem, which is mathematically formulated to specify the decision variables, the relationship among them and the constraints they must obey. Based on the proposed extended disjunctive graph model, the structural properties of PMJSS are thoroughly analysed. With exploiting these properties, a hybrid shifting bottleneck procedure (HSBP) algorithm combined with metaheuristics is developed for PMJSS. Extensive computational experiments are reported based on a set of benchmark JSS and PMJSS instances. To evaluate the optimality performance of the proposed methodology, a lower bound calculation method is specially proposed for PMJSS. The computational results indicate that the proposed HSBP algorithm can solve the PMJSS problem efficiently and effectively.

About academic contributions, the original-version SBP algorithm for JSS has been significantly improved to solve the PMJSS problem with some distinct novelties that are highlighted again as follows:

- detailed analysis of structural properties of PMJSS is given based on the extended disjunctive graph model and mathematical formulation.
- an innovative algorithm called the *topological-sequence algorithm* is proposed to decompose the PMJSS problem into a set of *single-machine scheduling* (SMS) and/or *parallel-machine scheduling* (PMS) subproblems with different release times and delivery times;
- the *modified Carlier algorithm* based on the proposed lemmas and the proofs is developed to solve the SMS subproblem;
- the *Jackson rule* is extended to quickly solve the PMS subproblem;
- a lower bound calculation method is developed for PMJSS in order to evaluate the optimality performance; and
- the *Tabu Search metaheuristic algorithm* is embedded under the framework of SBP to optimise the large-size JSS and PMJSS cases.

In fact, the proposed PMJSS model can be extended to be the *blocking parallel-machine job-shop scheduling* (BPMJSS) and *no-wait blocking parallel-machine job-shop scheduling* (NWBPMJSS) problems to identify, analyse, model and solve the real-world train scheduling problems (Liu and Kozan, 2009 and 2010). In a sense, the proposed methodology on PMJSS in this paper is an important and fundamental tool to analyse and solve the train scheduling problems.

References

Adams, J., Balas, E., & Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science, 34*(3), 391-401.

Alvarez-Valdes, R., Fuertes, A., Tamarit, J. M., Giménez**,** G.**,** & Ramos, R. (2005). A heuristic to schedule flexible job-shop in a glass factory. *European Journal of Operational Research 165*, 525–534.

Balas, W., & Vazacopoulos, A. (1998). Guided Local Search with Shifting Bottleneck for Job Shop Scheduling. *Management Science, 44*, 262–275.

Carlier, J. (1982). The one-machine sequencing problem. *European Journal of Operational Research, 11*, 42-47.

Chen, H., & Luh, P. B. (2003). An alternative framework to Lagrangian relaxation approach for job shop scheduling. *European Journal of Operational Research, 149*, 499-512.

Cheng, J., Karuno, Y., & Kise, H. (2001). A shifting bottleneck approach for a parallel-machine flowshop scheduling problem. *Journal of the Operations Research Society of Japan, 44*(2), 140-155.

Dauzère-Pérès, S., & Paulli, J. (1997). An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research, 70*, 281–306.

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

Gao, J., Sun, L., & Gen, M. (2008). A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers & Operations Research, 35*, 2892–2907.

Huang, W. Q., & Yin, A. H. (2004). An improved shifting bottleneck procedure for the job shop scheduling problem. *Computers & Operations Research, 31*, 2093-2110.

Ivens, P., & Lambrecht, M. (1996). Extending the shifting bottleneck procedure to real-life applications. *European Journal of Operational Research, 90*, 252-268.

Kellerer, H. (2005). Minimizing the maximum lateness. In J. Y-T. Leung (Ed.), *Handbook of Scheduling: Algorithms, Models, and Performance Analysis* (Chapter 10). USA: Chapman & Hall/CRC.

Liu, S. Q., & Kozan, E. (2010). Scheduling trains with priorities: a no-wait blocking parallel-machine job-shop scheduling model. *Transportation Science*, doi:10.1287/trsc.1100.0332.

Liu, S. Q., & Kozan, E. (2009). Scheduling trains as a blocking parallel-machine job shop scheduling problem. *Computers & Operations Research*, 36, 2840-2852.

Liu, S. Q., & Ong, H. L. (2004). Metaheuristics for the mixed shop scheduling problem. *Asia-Pacific Journal of Operational Research, 21*(4), 97-115.

Liu, S. Q., Ong, H. L., & Ng, K. M. (2005). A fast tabu search algorithm for the group shop scheduling problem. *Advances in Engineering Software, 36*, 533-539.

Mason, S. J., Fowler J. W., & Carlyle, W. M. (2002). A modified shifting bottleneck heuristic for minimizing total weighted tardiness in complex job shops. *Journal of Scheduling*, 5(3), 247–262.

Monch, L., & Drießel, R. (2005). A distributed shifting bottleneck heuristic for complex job shops. *Computers & Industrial Engineering, 49*, 363-380.

Monch, L., Schabacker, R., Pabst, D., & Fowler, J. W. (2007). Genetic algorithm-based subproblem solution procedures for a modified shifting bottleneck heuristic for complex job shops. *European Journal of Operational Research, 177*, 2100-2118.

Pfund, M. E., Balasubramanian, H., Fowler, J. W., Mason, S. J., & Rose, O. (2008). A multi-criteria approach for scheduling semiconductor wafer fabrication facilities. Journal of Scheduling, 11(29-47).

Ramudhin, A., & Marier, P. (1996). The generalized shifting bottleneck procedure. *European Journal of Operational Research, 93*, 34-48.

Sadeh, N., Sycara, K., & Xiong, Y. (1995). Backtracking techniques for the job shop scheduling constraint satisfaction problem. *Artificial Intelligence, 76*, 455-480.

Xia, W., & Wu, Z. (2005). An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Computers & Industrial Engineering, 48*, 409–425.

Appendix 1

Using a numerical example, Appendix 1 presents a proof that Theorem 2 proposed by Carlier (1982) may be incorrect. The data of a 2-job SMS example with different release times and delivery times is given in Table A1.

Table A1 The data of a 2-job SMS example with release times and delivery times

Job j	J_1	J_2
r_j (Release times)	10	12
p_j (Processing times)	3	3
q_j (Delivery times)	11	12

To enumerate all the solutions of this 2-job SMS example, the directed disjunctive graphs and the Gantt charts are drawn in Figure A1.

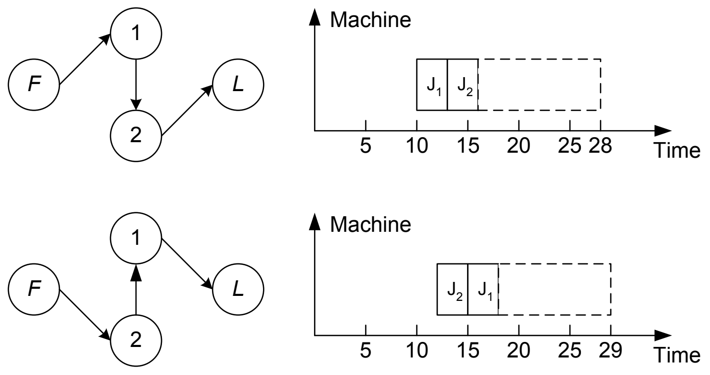

Fig. A1 The directed disjunctive graphs and Gantt charts for a 2-job SMS example

From Figure A1, it is evident that the optimal makespan is 28 and the optimal SMS sequence is {J1, J2}.

If {J1, J2} is a Schrage schedule, then we have a critical path $\{F,\ a, a+1, ..., c\ , L\}$ = {F, 1, 2, L}, critical sequence Λ= {1, 2}, interference job w = 1 as $q_1 < q_2$, critical subset $\Lambda_w = \{2\}$, and the makespan equal to 28.

According to Theorem 1 proposed by Carlier (1982), the value of $h(\Lambda_w)$ for this Schrage schedule is calculated as below:

$$h(\Lambda_w) = \min_{j\in\Lambda_w} r_j + \sum_{j\in\Lambda_w} p_j + \min_{j\in\Lambda_w} q_j = r_2 + p_2 + q_2 = 12+3+12 = 27$$

As this Schrage schedule is optimal, it is contradicted to Carlier's Theorem 2 because $h(\Lambda_w)$ equals 27 and the makespan ($C_{\max}$) of this schedule is 28, i.e. $h(\Lambda_w) \neq C_{\max}$ in this case.

Moreover, for all possible sets of jobs (i.e. $\{2\},\ \{1\},\ \{1,2\}, \{2,1\}$), we obtain

$$h(\Lambda_w) = h(\{2\}) = r_2 + p_2 + q_2 = 12+3+12 = 27$$
$$h(\Lambda_w) = h(\{1\}) = r_1 + p_1 + q_1 = 10+3+11 = 24$$
$$h(\Lambda_w) = h(\{1,2\}) = r_1 + p_1 + p_2 + q_1 = 10+3+3+11 = 27$$
$$h(\Lambda_w) = h(\{2,1\}) = r_1 + p_1 + p_2 + q_1 = 10+3+3+11 = 27$$

Obviously, there is no possibility of having a subset of jobs to satisfy Theorem 2 proposed by Carlier (1982).

Appendix 2

The proof for the proposed Lemmas I and II is given in Appendix 2. For any subset $A \subseteq J$ of jobs, the inequality is certain, thus we have

$$Opt \geq LB(A) \text{ and } LB(A) = \min_{j \in A} r_j + \sum_{j \in A} p_j + \min_{j \in A} q_j .$$

Assume $A = \Lambda$ and $\Lambda = \{a, a+1, ..., c\}$, we obtain,

$$LB(\Lambda) = r_a + \sum_{j=a}^{c} p_j + \min_{j \in \Lambda} q_j ,$$

due to $r_a = \min_{j \in \Lambda} r_j$ and there is no idle time between the processing of jobs a and c.

In addition, it is certain that,

$$C_{\max} = e_a + \sum_{j=a}^{c} p_j + q_c ,$$

where e_a is the starting time of job a. As $e_a \equiv r_a$, we obtain,

$$C_{\max} = r_a + \sum_{j=a}^{c} p_j + q_c .$$

Therefore, $C_{\max} = LB(\Lambda) + q_c - \min_{j \in \Lambda} q_j$.

Appendix 3

The proposed Lemma III is similar to Theorem 1 proposed by Carlier (1982). To complete analysing the properties of the Schrage schedule, here we present our own proof that is different from Carlier's proof.

As $\Lambda_w = \{w+1, ..., c\}$ is the subset of jobs processed after the interference job w, it is clear that $q_w < q_c \leq q_j$ and $e_w < r_j$ hold for all $j \in \Lambda_w$, where e_w is the starting time of the interference job w. Hence, inequality applied to $\Lambda_w = \{w+1, ..., c\}$ generates another lower bound:

$$LB(\Lambda_w) = \min_{j \in \Lambda_w} r_j + \sum_{j \in \Lambda_w} p_j + \min_{j \in \Lambda_w} q_j > e_w + \sum_{j \in \Lambda_w} p_j + q_c .$$

Since there is no idle time during the execution of the jobs in the critical sequence, the makespan of the Schrage schedule is $C_{\max} = e_w + p_w + \sum_{j \in \Lambda_w} p_j + q_c$.

Because $LB(\Lambda_w) > e_w + \sum_{j \in \Lambda_w} p_j + q_c$, we obtain

$$C_{\max} - LB(\Lambda_w) < C_{\max} - (e_w + \sum_{j \in \Lambda_w} p_j + q_c) \text{ and } C_{\max} - LB(\Lambda_w) < p_w .$$

Appendix 4

To verify the proposed Lemmas IV and V, a 7-job SMS example with different release times and delivery times introduced by Carlier (1982) is used for the following computational experiments. The data of this 7-job SMS example with different release times and delivery times is given in Table A2.

The following computational results will prove Lemmas IV and V.

Table A2 The data of a 7-job SMS example with release times and delivery times

Job j	J_1	J_2	J_3	J_4	J_5	J_6	J_7
r_j (Release times)	10	13	11	20	30	0	30
p_j (Processing times)	5	6	7	4	3	6	2
q_j (Delivery times)	7	26	24	21	8	17	0

Step 1:

After applying Schrage algorithm to this instance, the initial schedule is obtained and displayed in Table A3 and the makespan (i.e. the maximum completion-delivery time) can be obtained by

$$\begin{aligned} C_{\max} &= \min_{j\in\Lambda} e_j + \sum_{j=a}^{c} p_j + \min_{j\in\Lambda} l_j \\ . \quad &= \min\{10,15,21,28\} + (5+6+7+4) + \min\{38,32,25,21\} \\ &= 53 \end{aligned}$$

Table A3 The initial Schrage solution for one 7-job SMS problem

Results	J_1	J_2	J_3	J_4	J_5	J_6	J_7
Heads ($e_j \mid j \in J$)	10	15	21	28	32	0	35
Tails ($l_j \mid j \in J$)	38	32	25	21	8	43	0
Completion times	15	21	28	32	35	6	37
Completion-delivery times	53	53	53	53	43	49	37
Schrage Schedule	$J_6 \to J_1 \to J_2 \to J_3 \to J_4 \to J_5 \to J_7$						
Critical Sequence Λ	$J_1 \to J_2 \to J_3 \to J_4$						
Critical Job c	J_4						
Interference Job w	J_1 as $q_1 < q_4$						
Critical Subset Λ_w	$\{J_2, J_3, J_4\}$						
Makespan	53						

Step 2:

As the interference job w (Job 1) should be processed after the critical subset $\Lambda_w = \{2, 3, 4\}$, the release date of Job 1 is updated by:

$$r_w = \max\{r_w, \min_{j\in\Lambda_w} r_j + \sum_{j\in\Lambda_w} p_j\}$$
$$= \max\ \{10, \min\{13,\ 11,\ 20\}+(6+7+4)\ \}$$
$$= 28$$

After applying Schrage algorithm, a new Schrage solution is found and displayed in Table A4.

Table A4 The new Schrage solution after updating release date of Job 1

Results	***J₁***	***J₂***	***J₃***	***J₄***	***J₅***	***J₆***	***J₇***
Heads ($e_j \mid j\in J$)	28	18	11	24	33	0	36
Tails ($l_j \mid j\in J$)	11	26	32	21	8	39	0
Completion times	33	24	18	28	36	6	38
Completion-delivery times	44	50	50	49	44	45	38
Schrage Schedule	$J_6 \to J_3 \to J_2 \to J_4 \to J_1 \to J_5 \to J_7$						
Critical Sequence Λ	$J_3 \to J_2$						
Critical Job c	J_2						
Interference Job w	J_3 as $q_3 < q_2$						
Critical Subset Λ_w	$\{J_2\}$						
Makespan	50						

It is further observed from Table A4 that

$$C_{\max} = \min_{j\in\Lambda} e_j + \sum_{j=a}^{c} p_j + \min_{j\in\Lambda} l_j$$
$$= \min\{18,11\} + (6+7) + \min\{26,32\}$$
$$= 50$$

Step 3:

As the interference job w (job 3) should be processed after the critical subset $\Lambda_w = \{2\}$, we reset the release date of Job 3 by:

$$r_w = \max\{r_w, \min_{j\in\Lambda_w} r_j + \sum_{j\in\Lambda_w} p_j\}$$
$$= \max\ \{11,\ 13+6\} = 19$$

After reapplying the Schrage algorithm, a new Schrage solution is obtained and displayed in Table A5.

Table A5 The new Schrage solution after updating release date of Job 3

Results	***J₁***	***J₂***	***J₃***	***J₄***	***J₅***	***J₆***	***J₇***
Heads ($e_j \mid j\in J$)	33	13	19	26	30	0	38
Tails ($l_j \mid j\in J$)	7	32	25	21	12	38	0
Completion times	38	19	26	30	33	6	40
Completion-delivery times	45	51	51	51	45	44	30
Schrage Schedule	$J_6 \to J_2 \to J_3 \to J_4 \to J_5 \to J_1 \to J_7$						
Critical Sequence Λ	$J_2 \to J_3 \to J_4$						

Critical Job c	J_4
Interference Job w	ϕ as $q_4 = \min_{j\in\Lambda} q_j$
Critical Subset Λ_w	ϕ
Makespan	51

Lemma IV, "for a Schrage schedule, $C_{\max} = \min_{j\in\Lambda} e_j + \sum_{j=a}^{c} p_j + \min_{j\in\Lambda} q_j$ exists", is verified again by the fact that:

$$\begin{aligned} C_{\max} &= \min_{j\in\Lambda} e_j + \sum_{j=a}^{c} p_j + \min_{j\in\Lambda} l_j \\ &= \min\{13,19,26\} + (6+7+4) + \min\{32,25,21\} \\ &= 51 \end{aligned}$$

Note that the current schedule satisfies $q_c = \min_{j\in\Lambda} q_j$ for all $j \in \Lambda$ but its makespan is 51 that is not optimal. As a matter of fact, the optimal makespan is 50 with $q_3 < q_2$, $q_2 = q_c$ and $\Lambda = \{J3, J2\}$. Therefore, Lemma V ("Even if $q_c = \min_{j\in\Lambda} q_j$, this Schrage schedule may not be optimal") is proved. Lemma V indicates that the stopping condition (i.e. "*Schrage schedule is optimal if* $q_c = \min_{j\in\Lambda} q_j$, $\forall j \in \Lambda$") introduced in the book (Kellerer 2005) may be incorrect or incomplete. This is the main reason why we propose a modified Carlier algorithm for solving the SMS problem with different release times and delivery times.