Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2014, Article ID 431325, 9 pages
http://dx.doi.org/10.1155/2014/431325

Research Article

Variable Neighbourhood Search and Mathematical Programming for Just-in-Time Job-Shop Scheduling Problem

Sunxin Wang[1,2] and Yan Li[2]

[1] *School of Higher Vocational and Technical Education, Xi'an University of Technology, Xi'an 710082, China*
[2] *School of Mechanical and Precision Instrument Engineering, Xi'an University of Technology, Xi'an 710048, China*

Correspondence should be addressed to Sunxin Wang; wsx8280@126.com

Received 5 January 2014; Accepted 25 February 2014; Published 2 April 2014

Academic Editor: Gongnan Xie

This paper presents a combination of variable neighbourhood search and mathematical programming to minimize the sum of earliness and tardiness penalty costs of all operations for just-in-time job-shop scheduling problem (JITJSSP). Unlike classical E/T scheduling problem with each job having its earliness or tardiness penalty cost, each operation in this paper has its earliness and tardiness penalties, which are paid if the operation is completed before or after its due date. Our hybrid algorithm combines (i) a variable neighbourhood search procedure to explore the huge feasible solution spaces efficiently by alternating the *swap* and *insertion* neighbourhood structures and (ii) a mathematical programming model to optimize the completion times of the operations for a given solution in each iteration procedure. Additionally, a threshold accepting mechanism is proposed to diversify the local search of variable neighbourhood search. Computational results on the 72 benchmark instances show that our algorithm can obtain the best known solution for 40 problems, and the best known solutions for 33 problems are updated.

1. Introduction

The importance of "just-in-time" inventory management in industry has motivated the study of theoretical scheduling models. Among these models, many research efforts have been devoted to the classical earliness and tardiness (E/T) scheduling problems, and most of them consider earliness and tardiness penalties only for the last operation of the job [1–3]. Although the job may be finished on time, no storage and insurance cost are considered for intermediary operations which are processed as early as possible, which obviously violate the JIT philosophy [4].

For the just-in-time job-shop scheduling problem (JITJSSP), a job is decomposed in a sequence of operations that should be performed in a specific order. Each operation has its due date and earliness-tardiness penalties, which are paid if the operation is completed before or after its due date. Thus, the unwanted intermediate storage costs are avoided by finishing the operation as close to its due date as possible [4].

Baptiste et al. [4] defined an integer programming model for the JITJSSP problem and proposed methods based on two Lagrangian relaxations of the model to derive lower and upper bounds. They reported good lower and upper bounds for 72 instances. However, the gaps between lower and upper bound are still large for most instances. Monette et al. [5] later presented a constraint programming approach, which includes a novel filtering algorithm and dedicated heuristics. Although the method in [6] can give an inserted idle time schedule solution at last, it cannot offer the optimum (even near-optimum) solution for the JITJSSP with loose due date environment because few improvements occur on the instances of small scale.

In order to minimize the sum of total penalty costs in the JITJSSP problem, all operations are required to be completed as close to their due dates as possible, and thus the idle time is necessary to delay the start time of an early operation. In fact, it is a serious trouble how well idle time is inserted when severity resource contention occurs [7]. The mathematical programming method, which is suitable for a small solution space, can obtain an optimal solution. However, for the JITJSSP problem with a large number of jobs and machines, the computation time is intractability.

As for the JITJSSP problem with nonregular performance measure, how to explore the huge feasible space and how to insert machine idle time are two key issues. To deal with these two issues in solving the JITJSSP problem more effectively and efficiently, the hybrid algorithm is proposed in this paper with the mathematical programming (MP) model embedded to each iteration procedure of variable neighborhood search (VNS). In each iteration procedure of VNS, a sequence of operations on machines is produced by exploring the two swap and insertion neighborhoods of the current solution, and the mathematical programming model is then employed to insert idle time and optimize the completion times of the operations for the given sequences.

The rest of the paper is organized as follows. Section 2 discusses the problem formulation. The overall operational structure of the VNS/MP algorithm is presented in Section 3. An extensive variable neighborhood search study is discussed in Section 4. The mathematical programming model is presented in Section 5. A benchmarking study and experimental results are discussed in Section 6. Finally, the conclusions are given in Section 7.

2. Problem Formulation

The following definition of the JITJSSP problem is given on Baptiste et al. [4]. Consider a finite set of n jobs $J = \{J_1, J_2, \ldots, J_n\}$ that have to be processed on a finite set of m machines $M = \{M_1, M_2, \ldots, M_m\}$. Each job J_i has a sequence of m operations, $O_i = \{o_i^1, o_i^2, \ldots, o_i^m\}$, that need to be processed in sequence. The kth operation of job i, o_i^k, has to be processed by a prescribed machine $M(o_i^k) \in M$. Likewise, the set of operations that must be processed on each machine, $M_u \in M$, is denoted by $O(M_u)$. Without loss of generality, it is assumed that at most one operation of each job is processed on each machine, each machine can process at most one operation at a time, and no preemption is allowed. The notations used in the problem formulation are as follows:

p_i^k: processing time (duration) of operation o_i^k;

d_i^k: due date for operation o_i^k, assuming $d_i^k + p_i^{k+1} \leq d_i^{k+1}$;

st_i^k: start time of operation o_i^k;

c_i^k: completion time of operation o_i^k;

E_i^k: earliness of operation o_i^k, calculated by $\max(0, d_i^k - c_i^k)$;

T_i^k: tardiness of operation o_i^k, calculated by $\max(0, c_i^k - d_i^k)$;

α_i^k: earliness penalty per unit time for operation o_i^k;

β_i^k: tardiness penalty per unit time for operation o_i^k.

Object Function. We have

$$\text{Min} \quad \sum_{i=1}^{n}\sum_{k=1}^{m}\left(\alpha_i^k E_i^k + \beta_i^k T_i^k\right) \tag{1}$$

$$\text{Subject to} \quad c_i^{k+1} \geq c_i^k + p_i^{k+1}, \quad i = 1, 2, \ldots, n \wedge k = 1, 2, \ldots, m-1 \tag{2}$$

$$c_i^k \geq c_j^l + p_i^k \vee c_j^l \geq c_i^k + p_j^l, \quad \forall o_i^k, o_j^l \in O\left(M_u\right) \wedge i \neq j \tag{3}$$

$$\mathrm{st}_i^1 \geq 0, \quad i = 1, 2, \ldots, n. \tag{4}$$

The objective function (1) is to minimize the sum of earliness and tardiness costs of all operations. For each pair of consecutive operations o_i^k and o_i^{k+1} of the same job i, the precedence constraint (2) ensures that operation o_i^{k+1} cannot be processed before operation o_i^k is completed. The resource constraint (3) ensures that two operations of two different jobs i and j, o_i^k and o_j^l, in the set $O(M_u)$, cannot be processed simultaneously on machine M_u. Equation (4) ensures that the first operation of each is released after time zero.

3. The Overall Operational Structure of the VNS/MP Algorithm

Variable neighborhood search [8, 9] is a new local search technique that tries to escape from local optimum by changing neighborhood structures. In its basic form, VNS explores increasingly distant neighborhoods of the current incumbent solution, makes a local search from a neighbor solution to a local optimum, and jumps to it if and only if an improvement has been made.

The overall operational structure of the VNS/MP algorithm for the JITJSSP problem is shown in Figure 1. The VNS method is employed to explore the huge feasible space and provide a feasible schedule solution and, in initialization, shaking, and local search procedure of VNS. The mathematical programming model is introduced to make the operations complete closer to its due dates by inserting machine idle times into the given sequences and then feed backs an optimal schedule solution to VNS for its next iterative optimization procedure.

The overall steps of the VNS method are shown in Figure 1. Let N_k ($k = 0, 1, \ldots, k_{\max}-1$) denote neighbourhood structures and $N_k(x)$ the set of the solutions in the kth neighbourhood of x. To avoid consuming too much computational time, the maximum number of neighbourhood structures, $k_{\max}$, is suggested to be 2 [10]. The two kinds of neighbourhood structures, *swap* and *insertion*, are designed to change and improve the schedule solution for the shaking and local search procedure. Let N_0 indicate the neighbourhood *swap* and let N_1 indicate the neighbourhood *Insertion*. The stopping condition is maximum number of iterations N.

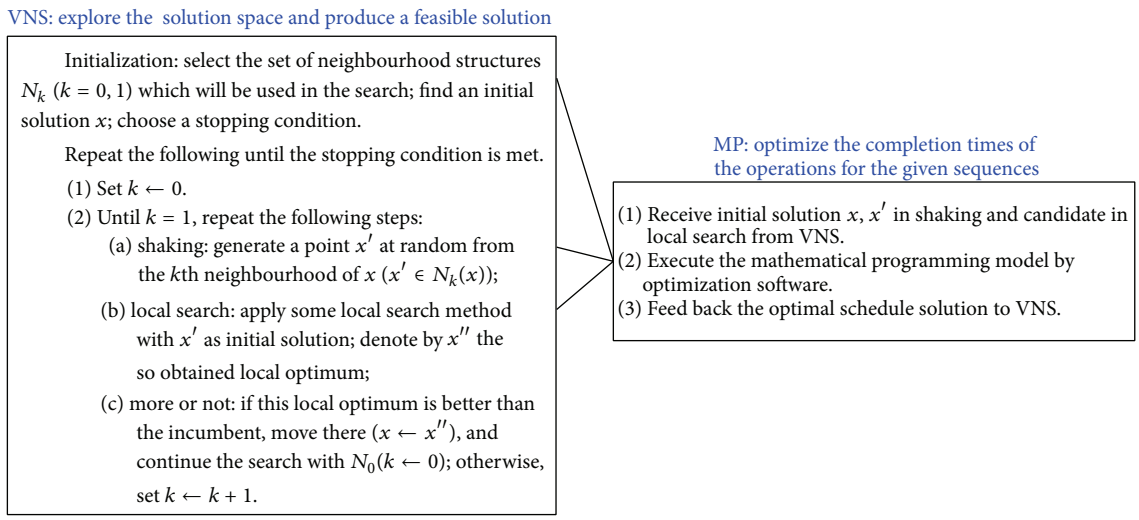

Figure 1: Overall operational structure of the VNS/MP algorithm.

4. Variable Neighbourhood Search: Explore the Solution Space

To achieve greater efficiency in VNS, we propose an excellent initial solution generation method and two kinds of neighborhood structures, *swap* and *insertion*, based on the characteristic of JITJSSP problem. Moreover, the local search procedure with the filter mechanism is designed to eliminate the swap movement that is unlikely to lead to good schedules. The key three issues in VNS, initialization, neighbourhood structures, and local search, are described in detail below.

4.1. Initialization. A good initial solution is able to reduce the computational time to achieve the optimal solution. Therefore, we present an initial solution generator that involves two new dispatching rules, each producing a candidate solution and the better one being selected as an initial solution. The rule, as a variant of the Earliest Due Date (EDD) rule, is that the operation with the earliest due date is selected first.

The other rule is a modification to ALL + CR + SPT [11]. Two separate sequences are produced for each machine by this new rule and the best solution is selected. The overdue jobs are placed in the priority sequence, in which the machine chooses the job whose p_i^k/β_i^k is minimal to be processed. But if this sequence is empty, the machine will choose the job whose p_i^k/α_i^k is maximal to be processed in the other sequence. For the criteria mean absolute lateness, ALL + CR + SPT can achieve the best performance at certain tightness levels. The new rule inherits this quality and can make the operation completion time closer to its due date.

4.2. Neighbourhood Structures: Swap and Insertion. Firstly, the *immediate job predecessor* and the *immediate job successor* of the operation u are denoted as JP[u] and JS[u], respectively. Similarly, the *immediate machine predecessor* and the *immediate machine successor* of u are denoted as MP[u] and MS[u], respectively. Besides, two successive operations u and v on the same machine (providing u is processed before v) are *adjacent* when there is no idle time between them (i.e., $c_u = \mathrm{st}_v$). And then u is the *adjacent machine predecessor* of v while v is the *adjacent machine successor* of u. Finally, a *block* is defined as a maximal sequence of size at least two, consisting of adjacent operations that are processed on the same machine.

The two neighbourhood structures, *swap* and *insertion*, are applied in shaking and local search procedures of VNS are described as follows.

4.2.1. Swap Neighbourhood. Swap the selected operation with its *adjacent machine successor* while it is early or when its *adjacent machine predecessor* is tardy.

In each iteration of VNS, the schedule solution obtained from mathematical programming model is the optimal solution for the current sequence, and the completion time of each operation is as close to its due date as possible. The first step of generating a neighbour in both neighbourhoods is to randomly select an operation which is either early and has an *adjacent machine successor* or tardy and has an *adjacent machine predecessor*. The selected operation by this *Swap* procedure has the two following lemmas.

Lemma 1. *If the selected operation u is early and has no adjacent machine successor (i.e., the immediate machine successor does not exist or there is idle time between u and the immediate machine successor), the immediate job successor of u is also early.*

Proof. If the selected operation u is early and has no *adjacent machine successor*, thus, $c_u < d_u$, where c_u and d_u denote the completion time and due date of u, respectively. c_u is equal to the start time of its *immediate job successor*; that is, $c_u = \mathrm{st}_{\mathrm{JS}[u]}$. There is $c_{\mathrm{JS}[u]} = \mathrm{st}_{\mathrm{JS}[u]} + p_{\mathrm{JS}[u]} = c_u + p_{\mathrm{JS}[u]} < d_u + p_{\mathrm{JS}[u]} \leq d_{\mathrm{JS}[u]}$, which means that the *immediate job successor* is also early. □

Lemma 2. *If the selected operation u is tardy and has no adjacent machine predecessor (i.e., the immediate machine predecessor does not exist or there is idle time between u*

(1) Get initial solution x', dr, and s.
(2) Set $q \leftarrow 0$ and $u \leftarrow 0$.
(3) While $q < q_{\max}$ do
  (a) $x'' \leftarrow$ the best one among the s candidates from $N_u(x')$.
  (b) If $f(x'') - f(x') \le dr$ then $x' \leftarrow x''$;
    else $u \leftarrow |u - 1|$.
  (c) $q \leftarrow q + 1$.
(4) $x'' \leftarrow x'$.

Algorithm 1: Overall operational structure of local search.

and the immediate machine predecessor), the immediate job predecessor of u is also tardy.

Proof. If the selected operation u is tardy and has no *adjacent machine predecessor*, the start time of u must be equal to the completion time of the *immediate job predecessor*; that is, $\mathrm{st}_u = c_{\mathrm{JP}[u]}$. Thus, $c_{\mathrm{JP}[u]} = \mathrm{st}_u > d_u - p_u \ge d_{\mathrm{JP}[u]}$, which means that the *immediate job predecessor* is also tardy. □

If the selected operation is early and has no *adjacent machine successor*, we will trace its job successors until finding one job successor which has an *adjacent machine successor*. If the job successor is early according to Lemma 1, then the job successor will be chosen to move. This *Swap* procedure is shown in Figure 2.

In Figure 2, let u be the selected operation; it is early and there is idle time between u and its *immediate machine successor* v. We trace the job successors of u, and the operation $\mathrm{JS}[u]$ meets exactly the swap condition. Thus, the operation $\mathrm{JS}[u]$ is to be swapped with the operation $\mathrm{JP}[v]$ to generate a new neighbour with good performance.

4.2.2. Insertion Neighbourhood. For block of a solution, it is not only the result of operations gathering together to compete for one machine but also an obstacle which prevents the operation from closing to its due date. Therefore, inserting the selected operation into the start or the end of the block can exploit the idle times effectively between the blocks for a performance improvement.

Let the selected operation be u, let the block which u belongs to be B, and let v (or w) denote the first (or last) operation of B. The *insertion* procedure varies with the status of u as follows.

Insertion Procedure When u Is Early. When the operation w is the last one of block B, move u to the right of w to bring new neighbour if $\mathrm{st}_w \le \mathrm{st}_{\mathrm{JS}[u]}$; otherwise, select an operation x randomly which is processed after u and move u to the right of w to bring new neighbour if $\mathrm{st}_x \le \mathrm{st}_{\mathrm{JS}[u]}$. See Figure 3.

Insertion Procedure When u Is Tardy. When the operation v is the last one of block B, move u to the left of v to bring new neighbour if $c_v \ge c_{\mathrm{JP}[u]}$; otherwise, select an operation x randomly which is processed before u and move u to the left of v to bring new neighbour if $c_x \ge c_{\mathrm{JP}[u]}$. See Figure 4.

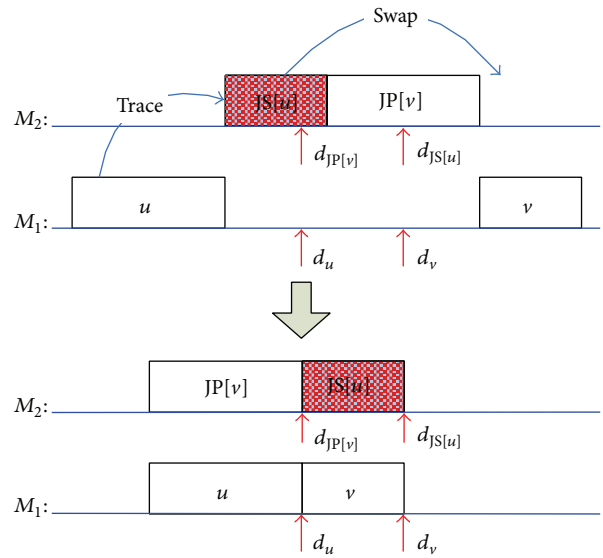

Figure 2: Example for *swap* procedure.

4.3. Local Search with the Filter Mechanism. As for the local search procedure in VNS, we make two improvements: threshold accepting mechanism to diversify the search space and filter mechanism to avoid the possible performance deterioration of schedule solutions in *swap* procedure.

4.3.1. Local Search with Threshold Accepting Mechanism. The threshold accepting mechanism [12] is embedded into each iteration process of local search procedure [13]. The current solution x' is replaced by the new solution x'' when the performance deviation between x' and x'' is less than a threshold dr. Set $dr = \sqrt{f(x')}/2$ in this paper. Furthermore, in order to enhance the search efficiency, a certain number s candidates are selected randomly from the set $N_k(x')$, and then let the best one in s candidates replace x''. See Algorithm 1.

4.3.2. Filter Mechanism for Swap Procedure. For a given sequence produced by VNS, an optimal schedule solution can be obtained by mathematical programming model. Thus, swapping two adjacent operations may delay the completion time of the successors and lead the new neighbour quality to deteriorate. In order to solve this problem, a filter mechanism is proposed in local search to prevent a new neighbour

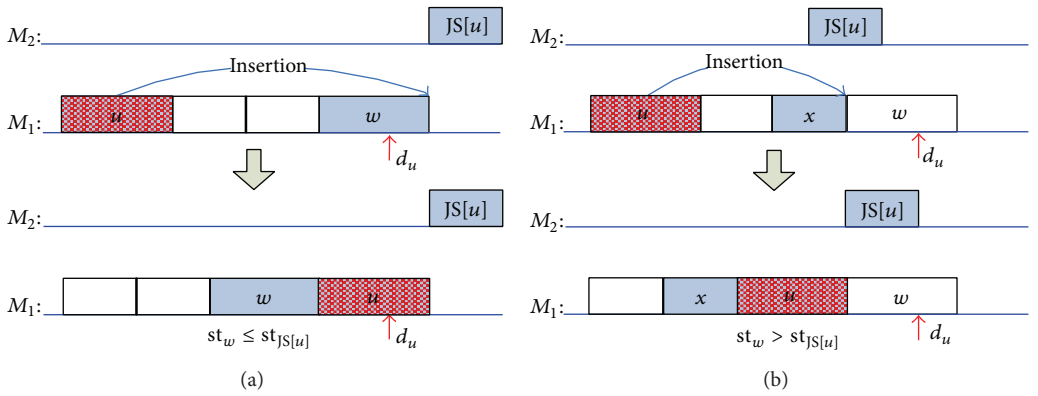

FIGURE 3: Insertion procedure for early operation.

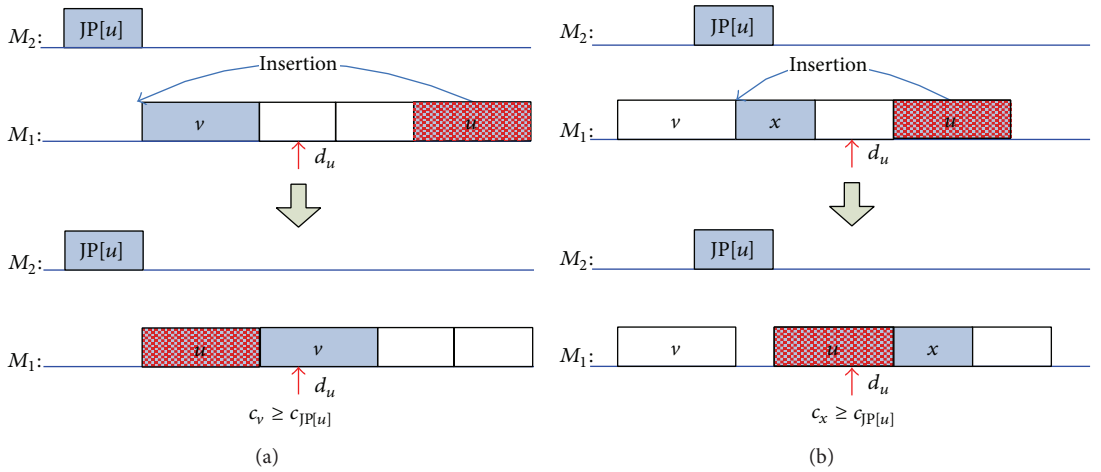

FIGURE 4: Insertion procedure for tardy operation.

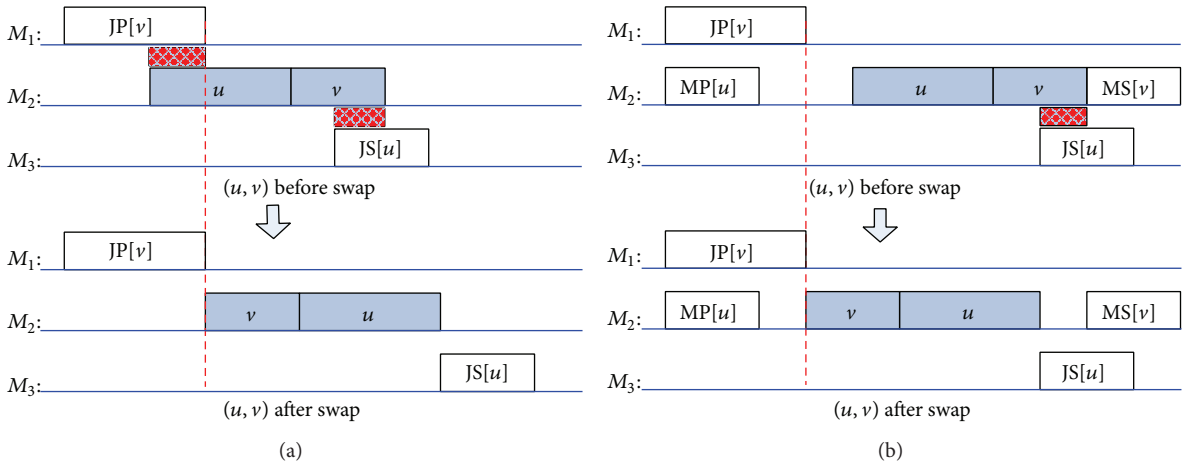

FIGURE 5: Overlap degree of two operations and its impact on swap procedure.

by *swap* operation from deteriorating the performance of schedule solutions.

Providing the adjacent operations to be swapped, u and v, we introduce *overlap* to evaluate effect of each *swap* operation. Figures 5(a) and 5(b) show two kinds of *overlap*.

In Figure 5(a), there is a large overlap between two operations, JP[v] and u, as well as JS[u] and v, before *swap* operation. However, the completion times of u and JS[u] are more seriously delayed once the *swap* operation is executed. On the other hand, in Figure 5(b), although there is also

an overlap between two operations, JP[v] and u, the *swap* operation can improve the solution because of the machine idle time.

In order to improve the solution and avoid the deterioration in *swap* operation, the overlap degree of two operations u and v is evaluated by (5), and the *swap* procedure can be executed if $\mathrm{Lap}(u, v) \leq \mathrm{Lap}_{\max}$. $\mathrm{Lap}_{\max}$ refers to a maximum of overlap degree:

$$\mathrm{Lap}(u, v) = \frac{\max\left(c_{\mathrm{JP}[v]}, c_{\mathrm{MP}[u]}\right) - \mathrm{st}_u + c_v - \min\left(\mathrm{st}_{\mathrm{JS}[u]}, \mathrm{st}_{\mathrm{MS}[v]}\right)}{p_u + p_v}. \quad (5)$$

5. Mathematical Programming Model: Optimize the Completion Times of the Operations for a Given Solution

For the VNS/MP algorithm, three steps in VNS procedure can produce three feasible solutions, respectively.

(1) Initialization step: an initial feasible solution is produced by EDD or ALL-JIT rule.

(2) Shaking step: a feasible solution is generated by *insertion* or *swap* operations.

(3) Local search step: a new neighbor and a given solution are also produced by *insertion* or *swap* operations.

All these three feasible solutions have satisfied the resource constraint (3), which can ensure that only one operation is processed simultaneously on one machine. Thus, by VNS procedure, the difficult resource constraint (3) is relaxed as follows:

$$\mathrm{st}_j^l + p_j^l \leq \mathrm{st}_i^k, \qquad M\left(o_i^k\right) = M\left(o_j^l\right), \quad (6)$$

where o_i^k is processed after o_j^l.

Therefore, in our proposed VNS/MP algorithm, VNS is used to explore the huge search space and achieve a good feasible solution. The resource constraint in JIT scheduling model is relaxed to make a good chance for the improvement by mathematical programming method. Thus, the JIT scheduling model can be changed into mathematical programming model, and the optimal completion times of all the operations for a given solution can be achieved in a short computation time.

6. Computational Results and Discussion

6.1. JITJSSP Instances. We used the same set of 72 instances generated by Baptiste et al. [4]. Each instance is named I-n-m-DD-W-ID, where I refers to "instance" and n and m are the number of jobs and machines, respectively, which have 9 combinations totally; that is, $(n \times m) \in (\{10, 15, 20\} \times \{2, 5, 10\})$. Each combination of n and m contains 8 instances such as DD, W, and ID which are chosen randomly.

(i) DD denotes the distances between due dates of consecutive operations which are exactly equal to the processing time of the last operation (DD = tight) or equal to the processing time plus a random value in [0, 10] (DD = loose).

(ii) W denotes the relation between earliness and tardiness penalty. There is W = equal if both α and β are chosen randomly in [0.1, 1] or W = tardy if α is chosen randomly in [0.1, 0.3] and β in [0.1, 1].

(iii) ID is set to be 1 or 2 to identify the two instances generated for each combination of the other parameters.

The VNS method is executed by a fixed number of $N = 100$ iterations. The maximum number of local search iterations is $q_{\max} = 50$. Set the number of the candidates and the upper bound of overlap to be $s = 3$ and $\mathrm{Lap}_{\max} = 0.5$, respectively.

6.2. Results and Comparison. The VNS/MP algorithm is coded in Matlab and the mathematical programming model is implemented by the Xpress optimizer in FICO Xpress.

The solution values for instances with $n = 10$, 15, and 20 jobs are reported, respectively, on Tables 1, 2, and 3. Table 4 shows the total number of best solutions for the 72 instances. The columns LB and UB list the best of the lower bounds and the upper bound reported by Baptiste et al. [4]. The column CP refers to the constraint programming approach proposed by Monette et al. [5]. The column EA/LS/MP refers to the combination of evolutionary algorithm, local search, and mathematical programming presented by Dos Santos et al. [6]. The best result of each instance computed by these five methods is highlighted in boldface type.

From Table 4, our VNS/MP algorithm obtains the best solution on 40 of the 72 instances and outperforms all of its competitors. Although VNS/MP has overwhelming dominance over the UB and CP, it is a little better than EA/LS/MP. Compared with EA/LS/MP, VNS/MP gives better solutions for most of the small instances; however, as the size of instances increases its performance deteriorates increasingly.

Yet we should not make a conclusion that VNS/MP is not suitable for solving JITJSSP problems of large scale. The neighborhood structures *swap* and *insertion* in VNS/MP are based on adjacent operations, which result in the solution space increasing along with the number of adjacent operations to be operated. As a consequence, the search space will become too huge when the due date setting is tight which makes most of the operations tend to be adjacent. Practically, most of the due dates of the 72 benchmark instances are extremely tight, which exactly coincides with the advantage of EA/LS/MP. In other words, VNS/MP solves the JIT scheduling problems with loose due dates effectively but its performance may decline when it is used to schedule problems with tight due dates.

7. Conclusions

We propose and evaluate a VNS/MP hybrid algorithm to the JITJSSP problem that searches a minimal sum of earliness and

Table 1: Results for instances with 10 jobs.

Instance	LB [4]	UB [4]	CP [5]	EA/LS/MP [6]	VNS/MP
I-10-2-tight-equal-1	434	**453**	461.96	463.58	461.96
I-10-2-tight-equal-2	418	458	**448.32**	**448.32**	**448.32**
I-10-5-tight-equal-1	660	826	764.80	765.35	**689.11**
I-10-5-tight-equal-2	612	848	779.40	779.52	**763.24**
I-10-10-tight-equal-1	1126	1439	1339.64	1281.66	**1277.44**
I-10-10-tight-equal-2	1535	2006	1902.30	2175.22	**1878.26**
I-10-2-loose-equal-1	219	225	**224.84**	**224.84**	**224.84**
I-10-2-loose-equal-2	313	324	**319.37**	331.08	**319.37**
I-10-5-loose-equal-1	1263	1905	1823.85	**1740.08**	1774.39
I-10-5-loose-equal-2	878	1010	999.14	1045.76	**967.73**
I-10-10-loose-equal-1	331	376	381.88	763.84	**364.39**
I-10-10-loose-equal-2	246	260	256.78	343.12	**249.85**
I-10-2-tight-tard-1	174	195	**179.46**	179.68	**179.46**
I-10-2-tight-tard-2	143	147	164.38	146.43	**145.37**
I-10-5-tight-tard-1	361	405	398.37	410.16	**387.30**
I-10-5-tight-tard-2	461	708	639.16	**632.91**	635.93
I-10-10-tight-tard-1	574	855	773.26	777.25	**687.65**
I-10-10-tight-tard-2	666	800	830.39	1031.13	**779.30**
I-10-2-loose-tard-1	416	**416**	416.44	416.44	416.44
I-10-2-loose-tard-2	137	138	**137.94**	143.00	**137.94**
I-10-5-loose-tard-1	168	188	182.64	259.02	**175.08**
I-10-5-loose-tard-2	355	572	513.91	509.33	**499.93**
I-10-10-loose-tard-1	356	409	387.05	438.04	**383.86**
I-10-10-loose-tard-2	138	152	**144.94**	169.07	**144.94**
Sum of best solutions		2	6	4	20

Table 2: Results for instances with 15 jobs.

Instance	LB [4]	UB [4]	CP [5]	EA/LS/MP [6]	VNS/MP
I-15-2-tight-equal-1	3316	3559	3641.19	**3344.54**	**3344.54**
I-15-2-tight-equal-2	1449	1579	1534.12	1480.97	**1479.76**
I-15-5-tight-equal-1	1052	1663	1504.04	**1369.78**	1402.99
I-15-5-tight-equal-2	1992	2989	2993.50	2728.02	**2693.24**
I-15-10-tight-equal-1	4389	8381	8189.70	**7340.27**	7395.36
I-15-10-tight-equal-2	3539	7039	5536.07	**5399.20**	6018.21
I-15-2-loose-equal-1	1032	1142	1249.68	**1041.70**	**1041.70**
I-15-2-loose-equal-2	490	520	524.10	506.44	**497.97**
I-15-5-loose-equal-1	2763	4408	3745.96	**3267.79**	3302.61
I-15-5-loose-equal-2	2818	4023	**3397.42**	3441.69	3760.00
I-15-10-loose-equal-1	758	1109	1033.06	1078.93	**986.43**
I-15-10-loose-equal-2	1242	2256	1792.67	1996.26	**1563.03**
I-15-2-tight-tard-1	786	913	835.52	790.61	**790.50**
I-15-2-tight-tard-2	886	956	947.17	905.73	**905.37**
I-15-5-tight-tard-1	1014	1538	1530.96	**1389.81**	1405.01
I-15-5-tight-tard-2	626	843	775.01	**712.50**	732.15
I-15-10-tight-tard-1	649	972	921.67	926.22	**813.46**
I-15-10-tight-tard-2	955	1656	1663.05	**1304.27**	1459.33
I-15-2-loose-tard-1	650	730	666.37	655.35	**654.84**
I-15-2-loose-tard-2	278	310	336.48	**291.68**	293.17
I-15-5-loose-tard-1	1098	1723	1478.97	**1315.53**	1396.54
I-15-5-loose-tard-2	314	**374**	401.65	396.14	396.06
I-15-10-loose-tard-1	258	312	300.11	356.26	**282.35**
I-15-10-loose-tard-2	476	855	658.90	**710.55**	778.40
Sum of best solutions		1	1	12	12

Table 3: Results for instances with 20 jobs.

Instance	LB [4]	UB [4]	CP [5]	EA/LS/MP [6]	VNS/MP
I-20-2-tight-equal-1	1901	2008	2115.58	**1940.30**	1941.18
I-20-2-tight-equal-2	912	1014	1104.20	948.86	**943.70**
I-20-5-tight-equal-1	2506	3090	3349.28	2933.28	**2853.31**
I-20-5-tight-equal-2	5817	7537	7883.50	**6915.06**	7038.92
I-20-10-tight-equal-1	6708	12951	14004.90	**10520.40**	11663.30
I-20-10-tight-equal-2	5705	9435	8535.88	7451.68	**7201.05**
I-20-2-loose-equal-1	2546	2708	2789.07	2551.60	**2550.53**
I-20-2-loose-equal-2	3013	3318	3386.88	**3109.29**	3213.52
I-20-5-loose-equal-1	6697	9697	9481.56	**7646.90**	8231.23
I-20-5-loose-equal-2	6017	8152	8835.72	**7294.50**	7642.51
I-20-10-loose-equal-1	3538	6732	6101.67	**5022.49**	5753.52
I-20-10-loose-equal-2	1344	2516	1963.05	1984.47	**1816.53**
I-20-2-tight-tard-1	1515	1913	1892.22	**1682.72**	1685.84
I-20-2-tight-tard-2	1375	1594	1704.26	1457.13	**1452.05**
I-20-5-tight-tard-1	3244	4147	4067.73	**3640.00**	3990.04
I-20-5-tight-tard-2	1633	1916	2040.70	**1873.80**	1911.91
I-20-10-tight-tard-1	3003	5968	5125.88	**4778.16**	5290.52
I-20-10-tight-tard-2	2740	3788	3938.51	**3270.09**	3491.52
I-20-2-loose-tard-1	1194	1271	1409.73	1206.97	**1204.92**
I-20-2-loose-tard-2	735	857	907.60	**774.22**	788.36
I-20-5-loose-tard-1	2524	3377	4015.62	**2973.23**	3720.83
I-20-5-loose-tard-2	3060	5014	4539.36	**3654.86**	3719.80
I-20-10-loose-tard-1	2462	6237	7287.00	**5100.46**	6169.68
I-20-10-loose-tard-2	1226	1830	1727.88	1634.58	**1588.70**
Sum of best solutions		0	0	16	8

Table 4: Total number of best solutions for the 72 instances.

Algorithm	UB [4]	CP [5]	EA/LS/MP [6]	VNS/MP
Total number of best solutions	3	7	32	40

tardiness penalty costs for all operations, not only for the jobs. This hybrid method combines (i) the variable neighbourhood search procedure to explore the huge feasible space of the JITJSSP problem efficiently by escaping from local optima with alternating the *swap* and *insertion* neighborhood structures and (ii) the mathematical programming model to insert idle machine time and optimize the completion times of the operations for a given sequence.

The VNS/MP hybrid algorithm is tested with 72 benchmark instances in JITJSSP problem and compared with three other approaches. It can find the best known solution for 40 instances with minimal computational effort, and the best known solutions for 33 instances are updated. As continuation to this work, we intend to design more efficiency neighborhood structures to improve the performance of VNS method to meet the large-scale JITJSSP problems with tight due date environment.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] K. R. Baker and G. D. Scudder, "Sequencing with earliness and tardiness penalties: a review," *Operations Research*, vol. 38, no. 1, pp. 22–36, 1990.

[2] Z. Zhu and R. B. Heady, "Minimizing the sum of earliness/tardiness in multi-machine scheduling: a mixed integer programming approach," *Computers and Industrial Engineering*, vol. 38, no. 2, pp. 297–305, 2000.

[3] S. Thiagarajan and C. Rajendran, "Scheduling in dynamic assembly job-shops to minimize the sum of weighted earliness, weighted tardiness and weighted flowtime of jobs," *Computers and Industrial Engineering*, vol. 49, no. 4, pp. 463–503, 2005.

[4] P. Baptiste, M. Flamini, and F. Sourd, "Lagrangian bounds for just-in-time job-shop scheduling," *Computers & Operations Research*, vol. 35, no. 3, pp. 906–915, 2008.

[5] J.-N. Monette, Y. Deville, and P. Van Hentenryck, "Just-in-time scheduling with constraint programming," in *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS '09)*, pp. 241–248, September 2009.

[6] A. G. Dos Santos, R. P. Araujo, and J. E. C. Arroyo, "A combination of evolutionary algorithm, mathematical programming, and a new local search procedure for the just-in-time job-shop scheduling problem," *Learning and Intelligent Optimization*, vol. 6073, pp. 10–24, 2010.

[7] J. J. Kanet and V. Sridharan, "Scheduling with inserted idle time: problem taxonomy and literature review," *Operations Research*, vol. 48, no. 1, pp. 99–110, 2000.

[8] P. Hansen and N. Mladenović, "Variable neighborhood search: principles and applications," *European Journal of Operational Research*, vol. 130, no. 3, pp. 449–467, 2001.

[9] W. Cheng, P. Guo, Z. Zhang, M. Zeng, and J. Liang, "Variable neighborhood search for parallel machines scheduling problem with step deteriorating jobs," *Mathematical Problems in Engineering*, vol. 2012, Article ID 928312, 20 pages, 2012.

[10] C.-J. Liao and C.-C. Cheng, "A variable neighborhood search for minimizing single machine weighted earliness and tardiness with common due date," *Computers and Industrial Engineering*, vol. 52, no. 4, pp. 404–413, 2007.

[11] T. C. E. Cheng and J. Jiang, "Job shop scheduling for missed due-date performance," *Computers and Industrial Engineering*, vol. 34, no. 2-4, pp. 297–307, 1998.

[12] V. Bouffard and J. A. Ferland, "Improving simulated annealing with variable neighborhood search to solve the resource-constrained scheduling problem," *Journal of Scheduling*, vol. 10, no. 6, pp. 375–386, 2007.

[13] M. Zandieh and M. A. Adibi, "Dynamic job shop scheduling using variable neighbourhood search," *International Journal of Production Research*, vol. 48, no. 8, pp. 2449–2458, 2010.