A Heuristic Algorithm for the Job Shop Scheduling Problem

Ai-Hua Yin

UFsoft School of Software Jiangxi University of Finance and Economics, Nanchang 330013, Jiangxi China
Aihuayin@mail.china.com

Abstract. The job shop scheduling problem that is concerned with minimizing makespan is discussed. A new heuristic algorithm that embeds an improved shifting bottleneck procedure into the Tabu Search (TS) technique is presented. This algorithm is different from the previous procedures, because the improved shifting bottleneck procedure is a new procedure for the problem, and the two remarkable strategies of intensification and diversification of TS are modified. In addition, a new kind of neighborhood structure is defined and the method for local search is different from the previous.
This algorithm has been tested on many common problem benchmarks with various sizes and levels of hardness and compared with several other algorithms. Computational experiments show that this algorithm is one of the most effective and efficient algorithms for the problem. Especially, it obtains a lower upbound for an instance with size of 50 jobs and 20 machines within a short period.

1. Introduction

The job shop scheduling problem with which we are concerned consists in scheduling a set of jobs on a set of machines for the objective of minimizing the make-span, i.e., the maximum of completion time needed for finishing all the jobs. Any scheduling is subject to the constrains that each job has a fixed processing order through the machines and each machine can process at most one job at a time.

The job shop scheduling problem is NP-hard in a strong sense and even is one of the hardest combinational optimization problems [5]. It is well known that only small size instances can be solved with a reasonable computational time by exact algorithms, however, for large size instances, some encouraging results have been recently obtained with heuristic algorithms that are based on local search method [1,8,14]. Generally, Starting from an initial feasible solution, a local search method iteratively selects a proper solution from the neighborhood. As the observation of Van Laarhoven et al. [17] and Nowicki et al. [18], both the choice of a good-initial solution and the neighborhood structure are important aspects of algorithm's performance.

This paper is a further research based on our recent work, and a heuristic algorithm that is based on a Tabu Search (TS) technology and on the improved shifting

H. Jin et al. (Eds.): NPC 2004, LNCS 3222, pp. 118-128, 2004.

bottleneck procedure (ISB) [9] is presented. Here, ISB is used to find a good-initial solution, and the local re-optimization procedure of ISB is used to direct the local search of TS from a region to some different one in the solution space. In the local search procedure of TS, we define a new kind of neighborhood structure that is different from the previous. These two points make certain of the efficiency and effectiveness of our algorithm.

In this paper, the job shop scheduling problem is formalized in terms of a mathematical model and is represented on a disjunctive graph. Then, the TS technique with two strategies of intensification and diversification are analyzed, and the new heuristic algorithm, denote TSISB, is described. Finally, computational results on several test problems instances are shown, and the algorithm is compared with some typical algorithms for the problem.

2. The Problem Definition

Let $J=\{1, 2, \ldots, n\}$ be a set of jobs, $M=\{1, 2, \ldots, m\}$ be a set of machines and $V=\{0, 1, 2, \ldots, N, \#\}$ be a set of operations. Each job consists of a sequence of operations each of which has to be processed on a given machine for a given time. Here, 0 and # represent the dummy *start* and *finish* operations, respectively. A schedule is an allocation of each operation to the time (start time) from which it is processed. In other words, it is an allocation of processing order of the operations on the machines. The problem is to find a schedule that minimizes the make-span, which is subject to constraints: (i) the precedence of operations on each job must be respected; (ii) once a machine starts to process an operation it can not be interrupted and each machine can process at most one operation at a time. Let A denote the set of pair of adjacent operations constrained by the precedence relations as in (i); V_k denote the set of operations that are processed by the machine k ($k \in M$); $E_k \subset V_k \times V_k$ be the set of pairs of operations which therefore have to be sequenced as specified in (ii); d_i and t_i be the process-time (fixed) and the start time (variable) of the operation i ($i \in V$), respectively. The process-times of both 0 and # are zero, i.e. $d_0 = d_\# = 0$. Now, the problem can be stated as following mathematic model:

$$
\begin{aligned}
&\textbf{min } t_\# \\
&t_i \geq 0 \qquad\qquad i \in V \\
&t_j - t_i \geq d_i \qquad (i,j) \in A \\
&t_j - t_i \geq d_i \vee t_i - t_j \geq d_j \qquad (i,j) \in E_k,\ k \in M
\end{aligned}
\tag{1}
$$

The first set of constraints means that $t = 0$ is the start time of the system, and the next two represent the constraints (i) and (ii), respectively, where "$\vee$" means "or". Any solution of (1) is called a schedule, a feasible solution of the problem.

It is useful to represent this problem on a disjunctive graph $G:=(V, A, E)$ [4], where V is the set of nodes, A is the set of ordinary (conjunctive) arcs and E is the set of disjunctive arcs. The node, the directed arc and the disjunctive pair-arc of G correspond to operation, precedence relation of two adjacent operations of a job and the pair-operation that are processed by the same machine, respectively. So, $E = \cup E_k$

($k \in M$), where E_k is the subset of disjunctive pair-arc corresponding to the pair-operation that are processed by the machine k. The weight (length) of each arc (i, j) is d_i that infers the process time of operation i, where $i \in V$, $(i, j) \in A \cup E$ and operation i is processed right before operation j.

Fig.1. is the disjunctive graph for an instance with $n = 3$, $m = 3$, and $N = 8$. The number of each conjunctive arc is the weight (length) of the arc and the weight of each disjunctive arc is removed.

A subset of E_k ($k \in M$) is called a selection S_k that contains just one arc of each disjunctive pair-arc of E_k, and S_k is acyclic if it doesn't contain any cycle. According to Adams [2], a feasible processing order of the operations on the machine k is equivalent to the only one acyclic selection S_k, and, to determine a processing order of the operations on a machine is to sequence this machine. So, to sequence machine k is to find an acyclic selection S_k of E_k.

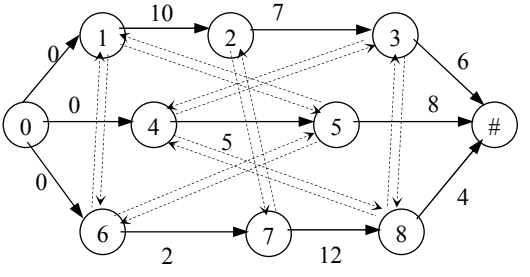

Fig. 1. Disjunctive graph of an instance

Let M_0 be the set of the machines that have been sequenced, so, a partial selection S is the union of selections S_k, one of each $E_k (k \in M_0)$. It is easy to understand that each S gives rise to a directed graph $D_S = (V, A \cup S)$. If D_S is acyclic, then S is acyclic, however, the converse is not true [2]. A complete selection S (i.e. $M_0 = M$) that generates an acyclic directed graph D_S defines a schedule, and it is a feasible solution of the job shop scheduling problem. To solve the problem is to find a complete selection S^* that gives rise to an acyclic D_{S^*} and minimizes the length of the longest (critical) path in D_{S^*}.

Let $G_k = (V_k, E_k)$, any acyclic selection S_k in E_k corresponds to the only one Hamilton path (denote H_k) of G_k, and the inverse is also true [2]. In this paper, the acyclic S_k, $S = \cup S_k (k \in M_0)$ and $D_S = (V, A \cup S)$ are replaced by the H_k, $H = \cup H_k (k \in M_0)$ and $D_H = (V, A \cup H)$, respectively.

For a feasible solution (a schedule) D_H, the swap of two adjacent operations processed by the same machine on the critical path may improve the solution [17], which is usually a base to define the neighborhood for local search. For this reason, the critical path is decomposed into a series of *critical blocks* ($B_1, B_2, \ldots, B_r$) [13]. Each block contains the operations processed on one machine, and any two operations, in the block B_i and $B_{i+1} (1 \le i < r)$, respectively, are processed by different machines.

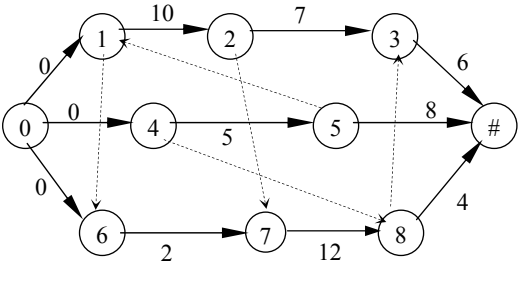

Fig. 2. A feasible solution of above instance

Fig.2. shows a feasible solution via a digraph D_H, H= ((5,1), (1,6))∪((2,7))∪((4,8), (8,3)). A critical path $P(0, \#)$ in this D_H is (0,4,5,1,6,7,8,3,#) with r=4, B_1=(4), B_2=(5,1,6), B_3=(7) and B_4=(8,3) and length of 47.

3. The Algorithm

Both the technique TS and the procedure ISB are the cornerstone of the new algorithm TSISB. In this section these two techniques are describes in details.

3.1 The Tabu Search

The Tabu Search technique proposed and formalized by Glover [10,11] is a meta-heuristic algorithm that is used to get optimal or near-optimal solution of combinational optimization problems. This method is based on an iterative procedure of neighborhood search, to find a member θ^* in a certain finite set Ω of feasible solutions, where θ^* minimizes some objective function $C(\bullet)$.

Neighborhood search methods are iterative procedures in which a neighborhood $N(\theta)$ must be predefined for each solution $\theta \in \Omega$, and each neighbor of θ is defined by some modifications of θ it-self. The next solution θ' to θ(one of the neighbors of θ) is searched among $N(\theta)$, and a step from θ to the θ' is usually called a *move*. Starting from a current feasible solution θ^c, all the neighbors in $N(\theta^c)$ are examined and the solution θ' with usually the best value of the objective function is chosen as the next solution, θ': $C(\theta') \le C(\theta'')$ $(\theta', \theta'' \in N(\theta))$. It is just the greedy scheme that is easy to get stuck of local optima. So, the strategy that the movement from θ^c to $\theta' \in N(\theta^c)$ is allowed even if $C(\theta') > C(\theta^c)$ helps the search escape from the trap of local optima. This strategy is one of the important characters of TS technology.

With TS scheme, the cycling, i.e. the search return to the solution that has been visited, may be met. To prevent this cycling, a structure called *Tabu list L* with length l is introduced in order to prevent the search from returning to a solution visited within the last l iterations. In general, the TS process stops when the $C(\theta)$ is close enough to the lower bound of $C(\bullet)$, or, when no improvement occurs over the best solution within a given number of iterations or the time-limit runs out.

3.2 Neighborhood Structure

It is known that there is no any fork on the searching track of TS. To this end, there must be more information to direct the exploration. In fact, there are *short-term* and *long-term* information concerned with the exploration process. This systematic use of the information is the essential feature of TS. The approach uses this strategy not only to avoid cycling but also to explore new directions in the neighborhood. The short-term information represented by the *Tabu list*, is based on the last l iterations and will partly prevent cycling. The long-term information contains C^*, the best value of $C(\bullet)$ found by TS so far.

The exploration process in Ω is described in terms of *move*. For each solution $\theta \in \Omega$, let $M(\theta)$ denotes the set of *moves* that can be applied to θ, and let a next solution of θ be $w = \theta \oplus v$, then the neighborhood of θ can be denoted as $N(\theta) = \{w \mid \exists v \in M(\theta), w = \theta \oplus v\}$. In general, the *move* is reversible, i.e. for each v there exits a move v^{-1} such that $(\theta \oplus v) \oplus v^{-1} = \theta$. So, instead of storing the information of complete solution, the *Tabu list* stores only the *move* it-self or the reverse of it associated with the *move* actually performed. Unfortunately, the restrictions of the *Tabu list* sometimes are so strong that they prevent getting a very good solution. This short-come can be overcome by using a sort of long-term information, *aspiration criterion* that allows the algorithm to choose a move from those forbidden moves, i.e. *tabu moves*. A *tabu move* applied to a solution θ is promising if it gives a solution better than the best one so far found.

The neighborhood structure used in our algorithm is described as following. In the next content, we denote the job-predecessor and job-successor of an operation w by $p(w)$ and $s(w)$, respectively. As a matter of fact, if $p(w)$ or $s(w)$ exists, then $(p(w), w)$ or $(w, s(w))$ belong to A. For a feasible solution D_H, denote $L(w, u)$ as the longest path from w to u in D_H, and $P(0, \#)$ as a critical path of D_H. Let $(w, h_1, h_2, \ldots, h_k)$ be a critical block of $P(0, \#)$, and $p(w)$ be in $P(0, \#)$. For any $h_i (i = 1, 2, \ldots, k)$, if there is

$$L(0,w) \geq L(0, p(h_i)) \tag{2}$$

then a *backward move* on w and h_i, i.e., let operation h_i be processed right before the operation w, will yield a new feasible solution [9]. This new solution is looked as one of neighbors of D_H. Also, let $(h_1, h_2, \ldots, h_l, u)$ be another critical block of $P(0, \#)$, and $s(u)$ be in $P(0, \#)$. For any $h_j (j = 1, 2, \ldots, l)$, if there is

$$L(0, s(h_j)) \geq L(0, u) \tag{3}$$

then a *forward move* on h_j and u, i.e., let operation h_j be processed right after the operation u, will yield a new feasible solution[9]. This new solution is looked as one of the neighbors of D_H, too. For all critical blocks of $P(0, \#)$, we test them by use of the inequalities (2) and (3) to generate all neighbors of D_H. It has been proved that swap w and h_1 or h_l and u must lead feasible neighbor of D_H, so the two inequalities are ignored in our algorithm.

Based on these two kinds of *moves*, the neighborhood of D_H consists of all those neighbors of it. Furthermore, this new kind neighborhood structure is different from all those used by previous authors, such as Larrhoven et al.[15], Nowicki et al.[16] and Pezzella et al.[17]. Compared with our neighbor- hood, those of Larrhoven and Pezzella are larger, which slows down the local search, and that of Nowicki is smaller, which usually limits the local search in a quite narrow area of the solution space. Balas takes use of a kind of neighborhood structure similar to ours, which gets a balance between the search speed and search space [5]. However, we reduce the computational cost in each step greatly.

Now, let θ^* be the best solution so far and I be the iteration counter, the TS procedure is described as follows:

Step1. Choose an initial solution θ, set $\theta^* = \theta$, $I = 0$;

Step2. $I = I+1$, and generates the subset $N^{\#}$ of $N(\theta)$ such that either the applied move does not belong to the Tabu list or at least one of the aspiration criterions satisfied;

Step3. Choose a best solution $\theta \in N^{\#}$ according to the objective function $C(\bullet)$ of the problem;

Step4. If $C(\theta) < C(\theta^*)$, then set $\theta^* = \theta$. Update the tabu list and the aspiration criterion;

Step5. If stopping criterion is met, then stop. Otherwise, go to **Step 2.**.

Where, Steps2, 3, 4 consists of the local search procedure of TS. Some of the stopping rules are as follows: □ $N(\theta)=\phi$; □ I is larger than the maximum number of iterations or the number of iterations is larger than a specific number since the last improvement of the best solution and □ the optimal or near-optimal solution is found.

3.3 Initial Solution and Tabu List

In most of the algorithm associated with TS technology, a good initial solution is fundamental for the computational performance of the algorithm. ISB is an affective algorithm for the job shop scheduling problem. This choice generation of the initial feasible solution allows our algorithm to obtain quite good solutions in comparable computational time or the same solution in shorter computational time.

The improved shifting bottleneck procedure is based on the famous SB that is proposed by Adams[2], and it solves one machine problem by DS (Schrage algorithm with disturbance). The main steps of ISB is as following:

Step1. Identify the bottleneck machine and sequence it with DS;

Step2. Re-optimize the machines of M_0 with DS in turn (at most 3 times), while keep the others fixed. If $M_0 = M$, stop. Otherwise, go to **Step1.**.

The procedure iterates over each machine and finishes when no improvement is found. At the last step, after the last machine has been sequenced, the procedure continues to local re-optimization until there is no improvement for the full cycle.

In the acyclic digraph D_H, let $r_i = L(0, i)$, $q_i = L(i, \#) - d_i$, DS is given as following:

Step1. Set $t = \mathbf{min}\{r_i;\ i \in V_k\}$, $R = V_k$;

Step2. If $r_i > t$, then $u_i = q_i - \delta(\ r_i - t)$, otherwise $u_i = q_i$, $i \in$ R;

Step3. Choose an operation from R, say j, with the greatest u_j, and if there are ties, break them by giving preference to the greatest q_j, and if there are ties still, break them by giving preference to the greatest d_j, and if there are ties other still, break them by choosing randomly. Set $t_j = \mathbf{max}\{r_j, t\}$, $R \Leftarrow R \setminus \{j\}$;

Step4. If $R = \phi$, stop. Otherwise, set $t = \mathbf{max}\{t_j + d_j, \mathbf{min}\{r_i;\ i \in R\}\}$, go to **Step 2.**.

Where δ is the disturbance coefficient with value of $\lceil 3\sqrt{n}/2 \rceil$, and n is the number of nodes in V_k. It is easy to know that the complexity of DS is $O(n^2)$.

As a matter of fact, the tabu list L is one of the components of the neighborhood structure of TS, and the value of the length l is an important parameter. Nowocki implements a fixed value of $l = 8$ [16], and Pezzella adopts a variable value of l from $2n/3$ to $2n$ [17]. Since any implementation of TS is problem oriented and needs particular definitions of values of tuning parameters such as l and level of aspiration [17], l is a semi-variable value of $\lfloor (n+m)/2 \rfloor$ in our experiments. Because of concerning

with n and m, this is a new way to determine the value of l. In TSISB, the way of updating the *Tabu list* is not the same as that of Nowocki's procedure. Especially, when $N^{\#}=\phi$, but $N(\theta^c) \neq \phi$, TSISB selects the oldest *tabu move* while not repeating the latest *tabu move*.

3.4 The Intensification and Diversification

Recently, TS is improved by *aspiration criteria, intensification* and *diversification* [12], which is to improve the effectiveness and efficiency of TS. However, our strategies of *intensification* and *diversification* are different from those of other authors.

Intensification strategy is to make the algorithm search around some smart solutions. TSISB implements this strategy not by back jumping scheme in the procedure of Nowocki et al. but by setting a quite large value of the up-bound of iterations. This up-bound is denoted as *Maxiter*. TSISB does not change the local search strategy of TS until it does not improve the best solution obtained so far within *Maxiter* iterations. Usually, the larger the *Maxiter* is the better the quality of solutions is. However, a larger *Maxiter* needs more computating time and the quality of solutions may not be improved indefinitely. In one word, the *intensification* procedure is just the local search procedure of TS, Steps 2,3,4 as described in §3.2.

On the other hand, *diversification* strategy is to make the algorithm search in different regions of the solution space, and these regions are far from each other. According to the literatures, the enough large number of iterations is used to differ these regions from each other [12]. To direct the search to different regions, TSISB implements the local re-optimization procedure of ISB after *Maxiter* steps of the local search of TS are implemented. This local re-optimization is the procedure of Step2 in ISB, where $M_0=M$, and it does not stop until there is no improvement during a full cycle. In fact, from a feasible solution, the re-optimization procedure is also a local search procedure whose neighborhood consists of the swap of any two operations processed by the same machine[5]. It is clear that the local re-optimization procedure with a low complexity is very different from the local search procedure of TS, which makes our *diversification* efficient and effective. In other words, once this strategy is implemented, usually the local search can really arrive at a new region. Further more, this *diversification* is different from those in literatures [3,6,7,17], one important reason is that the constraint of *tabu list* is ignored while implementing this *diversification* procedure. To get good solutions in a moderate period, the time of implementing diversification strategy is less than *Maxt*.

Let T denotes the time that this strategy is implemented, the main steps of TSISB is described as follows:

Step1. Get initial solution θ by ISB, and set $\theta^*=\theta$, $I=0$ and $T=0$;

Step2. $I=I+1$, and generates the subset $N^{\#}$ of $N(\theta)$ such that either the applied move does not belong to the tabu list or at least one of the aspiration criterions satisfied;

Step3. Choose a best solution $\theta \in N^{\#}$ according to the objective function $C(\bullet)$ of the problem;

Step4. If $C(\theta)<C(\theta^*)$, then set $\theta=\theta^*$. Update the Tabu list and the aspiration criterion;

Step5. If θ^* is optimal or equal to the lower bound, then stop. If $C(\theta) < C(\theta^*)$, set $I = 0$, and go to **Step2.**;

Step6. If $I < Maxiter$, go to **Step2.** . Otherwise, $T = T + 1$ and implement the re-optimization procedure;

Step7. If $T < Maxt$, set $I = 0$ and go to **Step2.**, Otherwise, stop.

4. Computational Results

TSISB is implemented in *C* language on personal computer Pentium 166MHz. The algorithm has been tested on 88 problem instances of various sizes and hardness level provided by OR-Library (http:// mscmga. Ms.ic.ac.uk/info.html) classed as following:

(a) Three instances FT6, FT10, FT20 due to Fisher and Thompson with $n \times m = 6 \times 6$, 10×10, 20×5, and five instances ABZ5-9 due to Adams et al. with two $n \times m = 10 \times 10$ and three $n \times m = 20 \times 15$.

(b) Eighty instances of eight different sizes ($n \times m$ = 15×15, 20×15, 20×20, 30×15, 30×20, 50×15, 50×20, 100×20) denoted as TD1-80. This class contains "partially hard " cases selected by Tailard among a large number of randomly generated instances [18]. The optimal solution is known only for 33 out of 80 instances.

TSISB is compared with all latest procedures for which we can find results (make-span, CPU time) in the literatures. The following notations are for those procedures: NS(3) gives the outcome of over three runs stand for the tabu search procedure of Nowicki and Smutniki; TD stands for the taboo search procedure of Taillard [18]. TSSB stands for a Tabu search procedure of Pezzella and Merelli [17]. SB-GLS1, SB-RGLS5,10 stand for three of the twelve guided local search procedures of Balas and Vazacopoulos[5]; and BV-*best* stands for the best solution of these 12 procedures.

Table 1. Comparison with TSSB on instances (a)

			OPT	**TSISB**			**TSSB**		
Problem	*n*	*m*	(UB LB)	UB	RE	Time	UB	RE	Time
FT6	6	6	55	55	0.00	-	55	0.00	-
FT10	10	10	930	930	0.00	200	930	0.00	80
FT20	20	5	1165	1165	0.00	73.2	1165	0.00	115
ABZ5	10	10	1234	1234	0.00	9.0	1234	0.00	75
ABZ6	10	10	943	943	0.00	231	943	0.00	80
ABZ7	20	15	656	665	1.37	2028	666	1.52	200
ABZ8	20	15	(645 669)	671	4.03	2196	678	5.12	205
ABZ9	20	15	(661 679)	686	3.64	2724	693	4.84	195
MRE					1.13			1.44	

The best lower bound (LB) for the problem is taken from [17]. The relative error RE (%) is calculated for each procedure and each instance, i.e. the percentage by which the solution obtained is above the LB, 100((UB-LB)/LB), and MRE means the mean relative error. The Time stands for computer independent CPU times that are based on Dongarra [7], as interpreted by Vaessens et al. [19].In our experiments, when $m \leq 15$ or $m > 15$, the parameters *Maxiter* gets the value 8000 and 12000 respectively; when $m \leq 10$, $m = 15$ or $m > 15$, *Maxt* gets the values 5, 10 and 15 respectively. The optimal solution and the lower bound for the stop criteria are equal to the LB.

Table1 compares TSISB with TSSB on instances (a). This class of instances includes the notorious FT10 (10×10) due to Fisher and Thompson , and it takes TSISB a quite reasonable period to obtain its optimal solution. Both of these algorithms find the optimal solution of five instances except three hard instances ABZ7,8,9, and the optimal solutions of ABZ8,9 are not known yet. Because the parameter *Maxt* is set as 10, TSISB makes greater efforts than TSSB to compute the three instances ABZ7-9, however, the MER of TSISB is smaller than that of TSSB.

Table 2. Comparison with other 7 algorithms on the instances of TD1-50

Problem Class	*n*	*m*	TD	NS(3)	SB-GLS1	SB-RGLS5	SB-RGLS10	BV-*best*	TSSB	TSISB
TD1-10	15	15	1.60	2.41	2.24	1.32	1.25	1.16	1.45	1.32
			–	(203)	(57)	–	–	(1498)	(2175)	(1097)
TD11-20	20	15	4.52	5.46	6.18	4.17	4.00	3.67	4.13	4.04
			–	(271)	(113)	–	–	(4559)	(2526)	(2232)
TD21-30	20	20	6.67	7.95	8.12	6.70	6.56	6.10	6.52	6.38
			–	(361)	(165)	–	–	(6850)	(34910)	(6644)
TD31-40	30	15	2.43	3.05	3.53	1.49	1.30	0.79	1.92	1.34
			–	(407)	(175)	–	–	(8491)	(14133)	(4101)
TD41-50	30	20	6.32	8.34	8.50	5.86	5.73	5.20	6.04	5.70
			–	(542)	(421)	–	–	(16018)	(11512)	(17784)
MRE			4.31	5.44	5.71	3.91	3.77	2.65	4.01	3.76

The average computing time for each class is in the parenthesis, – means not reported

Next, the 80 instances TD1-80 are computed. Among these instances, about 30 are easy because the number of jobs are several times that of machines [5], and the other 50 ones TD1-50 are hard not only because the number of their jobs and machines are almost same but also because their quite large sizes. The number of operations of these 50 instances is between 225 and 600, and these instances are divided into 5 classes according their sizes. Table2 gives the MRE for each of the class and the MRE of all these instances. Three of the 8 algorithms did not report their comput-

ing time. Not only on the MRE but also on the computing time, TSISB has the best performance, especially when the number of machines is equal to 20. It is contribute to several factors, such as, the producer for the initial solution, the *diversification* strategy and the new neighborhood in our algorithm.

TSISB has got the 28 optimal solutions out of the 30 easy instances except for TD62 and TD67 ($n \times m = 50 \times 20$) whose optimal solutions are not found yet. However, it obtains the best solutions of these two instances with the make-span of 2826 and 2879, respectively. Furthermore, the value of 2879 is the lowest up-bound so far of instance TD62 whose computing time is 8334 seconds, and the average computing time of TSISB on the 30 instances are much less than that of both TSSB and BV-*best*. In details, the average computing time for these three classes instances is TD51-60: 70.6 seconds; TD61-70: 2296 seconds and TD71-80: 186 seconds.

5. Conclusion

The new heuristic algorithm TSISB that is based on the TS technique and the improved shifting bottleneck procedure turns out to be effective and efficient. It gets initial solution with a new procedure ISB. TSISB implements TS procedure in a very nature way and improves the *intensification* and *diversification* strategies of TS by using both the local search procedure of TS and the local re-optimization procedure of ISB in turn. In the local search procedure of TS, TSISB adopts new neighborhood structures and executes the parameters δ, l, *Maxite* and *Maxt* in a simple manner.

The computational experiments show that TSISB is better than TSSB. TSISB performs better than SB-RGLS5,10 on the instances TD1-80. Especially, within a moderate period, TSISB has found a lower up-bound than them for the instance TD62 with a quite large size of $n \times m = 50 \times 20$. It can conclude that TSISB is robust, effective and efficient algorithm for its performance on the 88 benchmarks. Further more, other local search procedure and parallel algorithm can benefit from the way of implementing of *diversification* strategy in TSISB.

6. Reference

1. Aarts, E., LenstraJ, K.: Local Search and Combinational Optimization. Wiley, New York (1997)
2. Adams, J., Balas, E., Zawack, D.: The Shifting Bottleneck Procedure for Job Shop Scheduling. Management Sci. 3 (1988) 391-401
3. Applegate, D., Cook, W.: A Computational Study of Job-shop Scheduling. ORSA J. Computing. 2 (1991) 149-156
4. Balas, E.: Machine Sequencing via Disjunctive Graphs: An Implicit Enumeration Algorithm. Oper. Res. 17 (1969) 941-957
5. Balas, E., Vazacopoulos, A.: Guided Local Search with Shifting Bottleneck for Job Shop Scheduling. Management Sci. 2 (1998) 262-275

6. Dell'Amico, M., Trubian, M.: Applying Tabu-search to the Job-shop Scheduling Problem, Ann. Oper. Res. 4 (1993) 231-252
7. Dongarra, J. J.: Performance of Various Computers Using Standard Liner Equations Software. Report CS-89-85, Computer Science Department, University of Tennessee, Knoxville TN (1993)
8. French, S.: Sequencing and Scheduling: An Introduction to the Mathematics of the Job Shop. Wiley, New York (1982) 31-157
9. Huang, W.q., Yin, A.H.: An Improved Shifting Bottleneck Procedure for the Job Shop Scheduling Problem. Computers and Operations Research. 12 (2004) 2093-2110
10. Glover, F.: Tabu Search—Part I. ORSA J. Computing. 3 (1989) 190-206
11. Glover, F.: Tabu Search—Part II. ORSA J. Computing. 1 (1990) 4-32
12. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers Boston (1997)
13. Grabowski, J., Nowicki, E., Zdrzalaka, S.: A Block Approach for Single Machine Scheduling with Release Dates and Due Dates. European J. of Oper. Res. 1 (1986) 278-285
14. Hertz, A., Taillard, E., De Werra, D.: Local Search in Combinatorial Optimization. In: Aarts, E., Lenstra, J. (eds.): Tabu Search. vol. 5. Wiley, New York (1977) 121-136
15. Van Laarhoven, P. J. M., Aarts, E. H. L., Lenstra, J. K.: Job Shop Scheduling by Simulated Annealing. Oper. Res. 2 (1992) 113-125
16. Nowicki, E., Smutnicki, C.: A Fast Taboo Search Algorithm for the Job Shop Scheduling Problem. Management Sci. 6 (1996) 797-813
17. Pezzella, F., Merelli, E.: A Tabu Search Method Guided by Shifting Bottleneck for the Job Shop Scheduling Problem. European J. of Oper. Res. 2 (2000) 297-310
18. Taillard, E.: Benchmarks for Basic Scheduling Problems. European J. Operational Res. 4 (1993) 278-285
19. Vaessens, R. J. M., Aarts, E. H. L., Lenstra, J. K.: Job Shop Sheduling by Local Sarch. Memorandum COSOR 94-05. Eindhoven University of Technology, Department of Mathematics and Computing Science. Eindhoven The Netherlands (1994)

Aihua Yin: Ph.D.. UFsoft School of Software Jiangxi University of Finance and Economics. Research interests are in intelligent computing, combinatorial optimization, operational research, covering problem and SAT problem.