Available online at www.sciencedirect.com
ScienceDirect
Computers & Operations Research 35 (2008) 282–294
computers & operations research
www.elsevier.com/locate/cor

A very fast TS/SA algorithm for the job shop scheduling problem

Chao Yong Zhang*, PeiGen Li, YunQing Rao, ZaiLin Guan

School of Mechanical Science & Engineering, Huazhong University of Science & Technology, Wuhan 430074, PR China

Available online 3 April 2006

Abstract

The job shop scheduling problem (JSP) is one of the most notoriously intractable NP-complete optimization problems. Over the last 10–15 years, tabu search (TS) has emerged as an effective algorithmic approach for the JSP. However, the quality of solutions found by tabu search approach depends on the initial solution. To overcome this problem and provide a robust and efficient methodology for the JSP, the heuristics search approach combining simulated annealing (SA) and TS strategy is developed. The main principle of this approach is that SA is used to find the elite solutions inside big valley (BV) so that TS can re-intensify search from the promising solutions. This hybrid algorithm is tested on the standard benchmark sets and compared with the other approaches. The computational results show that the proposed algorithm could obtain the high-quality solutions within reasonable computing times. For example, 17 new upper bounds among the unsolved problems are found in a short time.

Keywords: Job shop scheduling problem; Tabu search; Simulated annealing; Makespan

1. Introduction

The job shop scheduling problem (JSP) with the makespan criterion comes from the manufacturing industry and has excellent practical applications. The problem can be described as follows: given n jobs that have to be processed on m machines. Each job consists of a predetermined sequence of task operations, each of which needs to be processed without interruption for a given period of time on a given machine. Tasks of the same job cannot be processed concurrently and each job must visit each machine exactly once. A schedule is an assignment of operations to time slots on a machine. The objective of the problem is to find a schedule that minimizes the makespan ($C_{\max}$), that is, the finish time of the last operation completed.

The job shop scheduling problem is widely acknowledged as one of the most difficult NP-complete problems [1]. This is illustrated by the fact that a relatively small instance with 10 jobs and 10 machines, proposed by Fisher and Thompson (1963) [2], remained unsolved for more than a quarter of a century, and until now no benchmark problem is solved to optimality for the 20×20 instances. During the past three decades, the problem has captured the interest of a significant number of researchers, and a lot of optimization algorithms and approximation algorithms have been proposed for solving it. The optimization algorithms, which are primarily based on the B& B scheme, have been successfully applied to solving small instances. However, they cannot accomplish optimal schedules in a reasonable

* Corresponding author. Tel.: +86 27 87556924(O); fax: +86 27 87543074.
E-mail address: superhust@163.com (C.Y. Zhang).

0305-0548/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2006.02.024

time for instances larger than 250 operations. On the other hand, approximation algorithms, such as priority dispatch, shifting bottleneck approach (SB) [3], genetic algorithm (GA) [4,5], simulated annealing (SA) [6,7], tabu search (TS) [8,9], would be very attractive alternatives for a large scale problem. Recently, most studies indicate that a single technique cannot solve this stubborn problem. Much work has been done on hybrid methods involving GA, TS, SA and SB techniques as hybrid methods can provide high-quality solutions within reasonable computing times. A comprehensive survey of scheduling techniques can be seen from Blażewicz et al. [10] and Jain and Meeran [11].

Among these approaches, TS and SA are two primary local search algorithms applied to solve the JSP. Tabu search initially proposed by Glover [12], which uses a memory function to avoid being trapped at a local minimum, currently seems to be one of the most promising methods for the job shop scheduling problem with the makespan criterion. However, tabu search was initially designed to find the near-optimum solution of combinatorial optimization problems and no clean proof of convergence is known. Like many local searches, the quality of the best solution found by tabu search approach depends on the initial solution. By contrast to TS, SA, which is analogous to the physical process of annealing [13], possesses a formal proof of convergence. Its behavior can be controlled by the cooling schedule and is not dependent on the initial solution. However, due to lack of the memory function, the searching may return to old solutions and become oscillation in local optimum surrounding. These greatly affect the effectiveness of SA and cause the search to consume excessive time. Therefore, the combination of the memory function (avoid cycling) of tabu search and the convergent characteristics of simulated annealing provide the rationale for developing a hybrid TS/SA strategy to solve the combinatorial optimization problems. In this paper, by reasonably combining the advantages of TS and SA, we develop an efficient hybrid optimization algorithm TSSA and investigate its potential on solving the JSP.

The remainder of this paper is organized as follows. Section 2 gives the framework of the hybrid of TS and SA for the JSP. In Section 3, the implementation of TSSA algorithm for the JSP is presented. In Section 4, we give the computational and comparative results on the benchmark instances. Conclusions are presented in Section 5.

2. The hybrid optimization strategy for the JSP

TS algorithm, defined and developed chiefly by Glover [14–16], seems to be an especially effective algorithm for the JSP. TS was firstly applied to the JSP by Taillard in 1989 [17]. Since then, researchers have introduced numerous improvements to Taillard's algorithm, and the most important contributions include those made by Nowicki and Smutnicki [9], Dell'Amico and Trubian [18], Barnes and Chambers [19] and Chambers and Barnes [20]. Among these TS approaches, algorithm TSAB developed by Nowicki and Smutnicki [9] introduces the significant breakthrough in both effectiveness and efficiency for the JSP. TSAB is perhaps the best known for its ability to locate an optimal solution for the notorious instance FT10 within only 30 s on a now-dated personal computer. However, even for the famous TSAB algorithm, the choice of an initialization procedure has an important influence on the best solution found, and a better initial solution might provide better results, see Jain et al. [21]. By contrast, SA is not a powerful technique for the JSP, but the initial solution has little influence on the solution quality obtained by SA procedure. Furthermore, hybrid SA approaches can provide strong competition to other methods [11]. Therefore, in this paper, by exploiting the properties of the JSP and the complementary strengths and weaknesses of the two paradigms, we present the newly designed hybrid TSSA algorithm in order to quickly obtain high-quality solutions and reduce the influence of the initial solution.

The idea of the hybrid approach for the JSP is based on the two important observations, due to Nowicki and Smutnicki [22], as follows. First, the solution space of the JSP owns BV and the best elite solutions dispersed over BV area. Second, TS is perfectly attracted to BV area. Even though the initial solution was set relatively far from the valley, elite solutions generated by TS can still be collected inside BV. The two observations indicate that TS is suitable for finding good solutions inside the BV, and these good solutions previously encountered provide better starting points for the further space exploration than various initial solutions do. However, the number of solutions inside BV is so large that it is unrealistic to expect that the whole valley might have been exhaustively searched. Nevertheless, SA, which possesses good convergence properties and accepts the candidate solution probabilistically by the *Metropolis acceptance criterion*, provides a procedure to find the sufficient "good" solutions over the big valley. Therefore, our hybrid TSSA algorithm, in which SA is used to find the promising elite solutions inside BV generated in the search history and TS intensifies search around these solutions, is proposed. The main idea of TSSA algorithm is also related

Step 1. Generate an initial solution s and calculate its makespan $f(s)$, set the current solution $s^* = s$, the best solution $s^b = s$, the best makespan $f(s^b) = f(s)$, $iter = 0$ and the initial temperature $T = t_0$, empty tabu list and push the s^* onto the elite solution stack L (LIFO list).

Step 2. Set $iter = iter+1$, generate neighbors of the current solution s^* by a neighborhood structure. If the s^* is optimal, then stop.

Step 3. Select the best neighbor which is not tabu or satisfies the aspiration criterion, and store it as the new current solution s^*. Update the tabu list.

Set the temperature T, and calculate the exact makespan $f(s^*)$.

If $(f(s^*) < f(s^b) \| \exp(f(s^b) - f(s^*))/T >$ random [0, 1])

{

"Push" the s^* onto the elite solution stack L.

}

Step 4. If $f(s^*) < f(s^b)$, then set $s^b = s^*$, $f(s^b) = f(s^*)$, $iter = 0$.

Step 5. If $iter \leqslant ImproveIter$ then go to Step 2.

Step 6. If a termination criterion is satisfied then stop. Else "pop" a solution on top of the solution stack L, shift the solution to active schedule and install the active solution as the current solution s^*, set $iter = 0$ and empty tabu list, then go to Step 2.

Fig. 1. Outline of TSSA algorithm for the JSP.

to the strategy that designs the more efficient forms of finite convergent TS based on recency-memory [23]. The general framework of hybrid TSSA algorithm for the JSP is outlined in Fig. 1.

In the hybrid TSSA algorithm, the core TS is a straightforward implementation of TS intensification strategy. A strong diversification strategy using SA procedure to find the elite solutions inside BV is equipped with the core TS and directs the intensified search to explore other regions of the solution space. More precisely, starting from a randomly initial solution, TSSA algorithm executes the core TS procedure and tracks the sufficiently "good" solutions found by SA procedure in the search history. The "good" solutions found by the SA procedure are stored in the elite solution stack L. Each new good solution is "pushed" onto the solution stack L when it is discovered. Subsequently, such solutions may be "popped" from the stack L in turn as new incumbent solutions, from which an intensified search is performed in a pre-specified number of iterations (*ImproveIter*). Given the suitable temperature T, the solutions in the elite solution stack should not be exhausted. The algorithm terminates when the total number of iterations reaches to a given value or the solution is proved to be optimal.

It can be seen from Fig. 1 that the hybrid TSSA framework can be converted to the traditional TS by setting the solution stack L size to 1 and omitting the SA unit. Such hybrid TSSA strategy retains the advantages of TS and SA, and provides a promising methodology to solve the other computationally intractable problems. For different problems, the neighborhood structure, algorithm criteria and parameters should be designed appropriately. Due to utilizing the properties of the JSP, the new hybrid algorithm has resemblance to TS. According to the characteristics of the different problems, it may even develop the different TS/SA framework, such as the SATS strategy which most closely resembles SA and incorporates TS into SA. In the next section, a detailed description of each component function of TSSA algorithm for the JSP is provided.

3. Implementation of TSSA algorithm for JSP

3.1. Initial solution

The initial solution can be generated by various methods such as the priority dispatching rules, the insertion algorithm, the shifting bottleneck procedure and random methods. Empirical study shows that the initial solution methods affect the solution quality obtained by TS algorithm, so that the better initial solution might provide better results [21]. Thus, for most of the TS algorithms, the specialized initialization procedures are used in order to obtain the better initial solution. However, the initial solution has little influence on the solution quality provided by TSSA algorithm. Therefore the search is initiated from the randomized active solution.

3.2. The neighborhood structure and move evaluation

A neighborhood structure is a mechanism which can obtain a new set of neighbor solutions by applying a small perturbation to a given solution [16]. Each neighbor solution is reached immediately from a given solution by a move. Neighborhood structures and move evaluation strategies are directly effective on the efficiency of the local search for the JSP, and unnecessary and infeasible moves must be eliminated if it is possible. Currently, the most well-known neighborhood structures are all based on the concept of blocks. A detailed survey of these neighborhood structures for the JSP can be seen by Blażewicz et al. [10] and Zhang et al. [24]. In the TSSA algorithm, taking into account a balance of the effectiveness and efficiency, for the general instances, N6[1] neighborhood structure introduced by Balas and Vazacopoulos [25] is adopted.

The running time of local search algorithm for the JSP is typically dominated by the cost of computing the makespans of neighbor solutions at each move. Therefore, some estimation approaches, which can quickly filter out moves that have a high probability of directing the search to the new elite solution, have been proposed to accelerate the search process [8,18,25]. Empirical testing shows that these estimation approaches not only significantly improve the efficiency of the search, but also have no material effect on the solution quality, see Jain et al. [21] and Zhang et al. [24]. In this paper, a remarkable estimation strategy suggested by Balas and Vazacopoulos [25] is implemented to perform these computations of each move. Moreover, when continuing the algorithm to the next iteration for the exact evaluation, there is no need to recalculate all head and tail values of the operations as many of them remain the same after the move. Therefore, in order to further reduce the computational cost, we only recalculate the head and tail values of the operations that need to be updated when exactly evaluating the selected neighbor solution at each step of the iteration.

3.3. Tabu list

The basic role of tabu list is to avoid the search process turning back to the solutions visited in the previous steps. The elements stored in the tabu list are the attributes of moves, rather than the attributes of solutions for the considered problem. The main purpose of using this attributive representation is to reduce the computational cost. However, a side effect of implementing the "a partial attribute" tabu list is that it may lead to giving a tabu status to an unvisited solution or even an interesting solution. The algorithm uses an "aspiration criterion", which accepts the move provided that its makespan is lower than that of the current best solution found so far, to avoid this problem. With our neighborhood structure, the move selected at each step may reverse more than one disjunctive arc and involve a sequence of operations. Unlike the majority of the previous TS for the JSP which only store the sequence of the operations exchanged in the tabu list, TSSA algorithm stores not only the sequence of the operations, but also their positions on the machine. This approach could better represent the attributes of moves. More precisely, if a move consists of the exchange on operations u and v, then a move achieved the same sequence of operations and positions ((namely, the operations from u to v is $(u, \ldots, w, \ldots, v)$ and their positions on the machine from u to v is $(p_u, \ldots, p_w, \ldots, p_v)$) is not permitted for the duration that the move is recorded as "tabu".

The length of tabu list determines the time limit of elements remaining on memory, which is discouraged at the current iterations. Therefore, it has an important influence on the course of the search. If the length of tabu list is too short, cycling cannot be avoided; conversely a too long size creates too many restrictions. However, given an optimization problem it is often difficult or even impossible to find a value that prevents cycling and at the same time does not excessively restrict the search for all instances of the problem of a given size. An effective approach for circumventing this difficulty is to use a dynamic tabu list size [15]. Empirical studies show that it may be possible to obtain superior results to allow a tabu list with variable size during the course of the search [8]. Therefore, in the TSSA algorithm, dynamic tabu list is applied and the length of tabu list is randomly chosen between two given minimal and maximal values $[L_{\min}, L_{\max}]$. Our preliminary experiments show that the suitable length of tabu list increases as the ratio of the number of jobs (n) to the number of machines (m) becomes larger. The smallest length of tabu list could be set to $L = 10 + n/m$ for getting good results, and $L_{\min} = [L]$, $L_{\max} = [L + 2]$ are appropriate values for the considered problem.

[1] The notation N6 and the following N1 are taken from Blażewicz et al. [10].

3.4. The recovery of the elite solutions based on the SA

In this paper, a powerful recency-based memory mechanism, which utilizes SA to find the elite solutions inside BV, is built in the core TS and induces the search to pursue different trajectories. The recency-based memory mechanism is adjusted as follows. If the current solution satisfies: $f(s^*) < f(s^b)$ or probability $\exp(f(s^b) - f(s^*))/T >$ random [0, 1] ($f(s^*)$ and $f(s^b)$ stand for the makespans of the current solution and the best solution, respectively), then the solution is pushed onto the elite solution stack L. However, if a pre-specified number of iterations (*ImproveIter*) have been executed without an improvement in the best-so-far solution, the solution on top of the solution stack L is popped, shifted to the active solution and the active solution is installed as the current solution. The TSSA algorithm then reinitiates the core TS procedure from the new current solution, as well as resets the original parameters and clears the tabu list.

During the run of the TSSA, the elite solutions found by SA are stored in the elite solution stack L. The stack L is maintained as a circular linked list which is pushed (popped) by allowing a list pointer to advance; the advancing list pointer overwrites the oldest entries when the solution stack L has been full stored. The maximum number of the solution stack L is a fixed number denoted as *Maxelite*. For the test instances, a *Maxelite* of 30 is found to offer an appropriate value. Temperature T of SA has an important influence on the selected elite solutions and consequently affects the quality of solution provided by TSSA algorithm. If the temperature is too low, the algorithm may be terminated earlier due to the elite solutions being quickly exhausted, whereas if the temperature is too high, it can not guarantee that the "effective" elite solutions are selected. Empirical testing shows that the suitable temperature T increases as the instances size becomes larger. For the general JSP instances, T can be set to value 2–6 according to the instance size. It can be set as $T = bestMakespan/Temp$, where *bestMakespan* is the best makespan found so far and *Temp* means a fixed parameter based on the instance size. It can be seen that the temperature decreases as the best makespans being found, so SA performs a "fine" search around the local optima. In addition, it must be noted that the solutions that run during the course of the tabu search only guarantee the semi-active schedules, not the active schedules. However, the optimal schedule is in the set of the active schedules. Therefore, TSSA algorithm shifts the semi-active schedules popped from the elite solution stack to active schedules. This approach could better direct the search to explore new promising regions and hopefully increase the chances of finding the global optimum.

3.5. Move selection

The choice rule of the TS method is to select the move which is non tabu with the lowest makespan or satisfies the aspiration criterion. Nevertheless, a situation may arise where all possible moves are tabu and none of them can satisfy the aspiration criterion. In such a case, one might use the oldest tabu move or randomly select a tabu move. Empirically, the second strategy that randomly selects a move among the possible moves proves better and is implemented in the TSSA algorithm.

3.6. Cycle check

TSSA algorithm employs a simple and fast mechanism similar to TSAB, with the exception of setting the *cycle_gap* to fixed numbers, to detect the cycling behavior. The detailed contents can be seen by Nowicki and Smutnicki [9] and Jain et al. [21]. When a cycle is detected, instead of continuously intensifying search to the current solution by the N6 neighborhood, we apply the N1 neighborhood structure to yielding a small perturbation to the current solution. This mechanism is able to remain nearby the current solution while simultaneously inducing the search to escape from the cycling.

3.7. Termination criterion

The algorithm stops when the algorithm has performed a given total number of iterations (*TotIter*), or the elite solution stack has been exhausted, or the solution is proved to be optimal. If one of the following conditions is satisfied: (1) All critical operations are processed on the same machine (i.e. only one critical block is generated) or belong to the same job (i.e. each block consists of only one operation); (2) The makespan is equal to the known lower bound, then

the solution is optimal and the algorithm is terminated. In addition, the elite solution stack should not be exhausted provided that the temperature T is suitably set.

4. Computational results

The proposed TSSA algorithm was implemented in VC++ language on a personal computer PentiumIV3.0G. In order to evaluate and compare the performance of TSSA algorithm, we tested on the 162 well-known benchmark problems taken from literature. These job shop scheduling instances include the following classes:

(a) Three instances denoted as FT6, FT10, FT20 (size $n \times m = 6 \times 6,\ 10 \times 10, 20 \times 5$) due to Fisher and Thompson [2], 40 instances LA01-40 (size $n \times m = 10 \times 5, 15 \times 5, 20 \times 5, 10 \times 10, 15 \times 10, 20 \times 10, 30 \times 10, 15 \times 15$) due to Lawrence [26], five instances ABZ5-9 (size $n \times m = 10 \times 10, 20 \times 15$) due to Adams et al. [3] and 10 instances ORB01-10 (size $n \times m = 10 \times 10$) due to Applegate and Cook [27].
(b) Four instances denoted as YN1-4 (size $n \times m = 20 \times 20$) due to Yamada and Nakano [28] and 20 instances SWV01-20 (size $n \times m = 20 \times 10, 20 \times 15, 50 \times 10, 50 \times 10$) due to Storer et al. [29].
(c) Eighty instances denoted as TA01-80 (size $n \times m = 15 \times 15, 20 \times 15, 20 \times 20, 30 \times 15, 30 \times 20, 50 \times 15, 50 \times 20, 100 \times 20$) due to Taillard [30].

The FT, LA, ABZ, ORB, YN and SWV problems are available from OR-Library,[2] while the TA problems are available from Taillard's web site [31]. For these benchmark sets, the best known upper bounds (UB_{best}) and the best known lower bounds (LB_{best}) are taken from Jain and Meeran [1111] and updated with the improved results from Nowicki and Smutnicki [32] and Taillard's web site [31].

We compared the TSSA algorithm with the best approximation algorithms which provide the detailed computational results, and used the following notation for those algorithms: TSAB stands for the TS of Nowicki and Smutnicki [9], BV stands for the guided local search with Shifting Bottleneck of Balas and Vazacopoulos [25]. Meanwhile, SB-RGLS5 stands for the solution of SB-RGLS5 procedure of Balas and Vazacopoulos and BV-best stands for the best solution obtained by Balas and Vazacopoulos. TSSB stands for a tabu search method guided by shifting bottleneck of Pezzella and Merelli [33] and *i*-TSAB stands for the tabu search algorithm of Nowicki and Smutnicki [32]. Among these papers listed above, the algorithms TSAB, BV and TSSB provide the detailed makespan and running time of each instance. But for the algorithm *i*-TSAB, which is an extension of their earlier TSAB algorithm, we can only obtain the results and running time of each group.

To analyze the quality of the solutions, the mean relative error (MRE) was calculated from the best known lower bound LB_{best}, and the upper bound (UB_{solve}) that is the makespan of the best solution solved by our algorithm, using the "relative deviation" formula $RE = 100 \times (UB_{solve} - LB_{best})/LB_{best}$ for each instance. During various tests (with tuning parameters) and standard tests, TSSA algorithm found many new upper bounds. The best upper bounds may be found by the run of many times, but standard tests only ran 10 times to get the average makespan and running time of each instance. Due to the stochastic properties of this algorithm, it is not reasonable to compare the b-MRE (best makespan MRE) of TSSA with the results of the other algorithms. Therefore, in order to fairly evaluate the performance of TSSA algorithm, we compared the mean performance (av-MRE) of TSSA with the results of the other algorithms.

TSSA algorithm offers a very short running time within several minutes (even seconds) on our personal PC for the general hard instances, and it requires about ten minutes for the particularly hard instances. Moreover, empirical studies of processor speeds show that it is hard to get the real computer-independent CPU time. Hence, in this paper we enclosed for each algorithm the original name of machine and the original running time to avoid discussion about the different computers speed used in tests.

4.1. Parameter set for experimental instances

TSSA algorithm was initiated from the active solution randomly generated. Parameter the length of tabu list, *ImproveIter* (the umimproved iterations), T (temperature) and *TotIter* (the total number of the iterations) were chosen

[2] The anonymous ftp site: ftp://mscmga.ms.ic.ac.uk/pub/jobshop1.txt. These instances are downloaded in January, 2004.

Table 1
Comparison with the other three algorithms for LA and ABZ instances

Problem group	Size	TSSA			TSAB[a]		BV-best[b]		TSSB[c]	
		b-MRE	av-MRE	T_{av}(s)	MRE	T_{av}(s)	b-MRE	T_{av}(s)	MRE	T_{av}(s)
LA01-05	10 × 5	0.00	0.00	0.0	0.00	3.8	0.00	3.9	0.00	9.8
LA16-20	10 × 10	0.00	0.00	0.2	0.02	68.8	0.00	25.1	0.00	61.5
LA21-25	15 × 10	0.00	0.03	13.6	0.10	74	0.00	314.6	0.10	115
LA26-30	20 × 10	0.02	0.02	15.2	0.16	136.4	0.09	100.0	0.46	105
LA36-40	15 × 15	0.03	0.19	36.1	0.28	375.6	0.03	623.5	0.58	141
ABZ5-6	10 × 10	0.00	0.00	2.7	0.08	16.5	0.00	252.5	0.00	77.5
ABZ7-9	20 × 15	2.10	2.80	88.9	4.34	–	2.45	6680.3	3.83	200
Average		0.31	0.43	22.3	0.71	–	0.37	1142.8	0.71	101.4

[a]The CPU time on the personal computer AT386DX.
[b]The CPU time on the SUN Sparc-330.
[c]The CPU time on personal computer Pentium 133 MHz.

experimentally in order to ensure a compromise between efficiency and effectiveness. The choice of the length of tabu list can be seen in Section 3.3. In this paper, the other parameter values are set as follows:

Parameter *ImproveIter* was bounded by the given minimal and maximal values between 2500–5000. More precisely, if the result of $10 \times n \times m$ is between 2500–5000, then $ImproveIter = 10 \times n \times m$. Otherwise, if the result of $10 \times n \times m$ is less than 2500 or more than 5000, then $ImproveIter = 2500$ or $ImproveIter = 5000$, respectively. For the test instances, we set $T = bestMakespan/Temp$ (see Section 3.4). Meanwhile, if the number of operations is less than or equal to 400, $Temp = 300$. Otherwise, $Temp = 300 + 50 \times n/m$. The parameter *TotIter* has an influence on the running time and the solution quality of this scheme. The increasing of *TotIter* yields a higher possibility of obtaining a high-quality solution. However, a further increase of this parameter has little influence on the makespan but evidently increases the running time. In order to ensure the balance between the running time and the solution quality, our preliminary experiments show that *TotIter* can be set to $n \times 100\,000/2$ for general hard instances, but for the particularly hard instances it increases correspondingly for the sake of finding a better solution.

4.2. Results for instances (a)

In this section we discuss the behavior of TSSA on the four oldest benchmarks: FT6,10, 20, ABZ5-9, LA01-40 and ORB01-10. The number of their operations ranges from 55 to 300. Despite their relatively small size, these instances were very hard to solve. For example, FT10 remained unsolved until twenty years later. However, by years, these instances have been solved optimally except for ABZ8 and ABZ9, some of them by the B& B scheme, some by approximate algorithms.

Firstly, the common benchmarks FT, LA and ABZ are tested by TSSA algorithm. The problems FT20(20×5), LA01-05(10×5), LA6-10(15×5), LA11-15(20×5) and LA30-35(30×10) are relatively easy because the number of jobs is several times larger than the number of machines. They could be solved to optimality by TSSA algorithm in a second, so their results are omitted from the table. Table 1 shows the comparison of the performance of TSSA algorithm with those of TSAB, BV-best and TSSB. This table, lists the best MRE (b-MRE), the average MRE (av-MRE) and the average running time of each group (T_{av}) of each algorithm. We enclose the original running time and original machines reported by the authors of TSAB, BV-best and TSSB in Table 1 (similar to Tables 2–7 below). It can be seen that TSSA algorithm performs very quickly and obtains the results in only half minute on average on the personal PC. On these problems, the av-MRE of TSSA is clearly lower than the MRE of TSAB and TSSB, and the b-MRE of TSSA is also better than the b-MRE of BV-best.

To make a more detailed performance comparison of the TSSA algorithm with the other algorithms, we select the 15 most difficult instances among FT, LA and ABZ benchmarks. The majority of the fifteen instances have been viewed as computational challenges, and even the optimal solutions of the instances ABZ8 and ABZ9 remain unknown until now. Table 2 shows the makespan performance statistics of each algorithm for the fifteen difficult problems. In this table, the column named UB(LB) lists the best known upper bounds (lower bounds) indicated in Jain and

Table 2
Results for the fifteen tough instances

Problem	Size	UB(LB)	TSSA			TSAB	BV		TSSB
			Best	M_{av}	T_{av}(s)		SB-RGLS5	BV-best	
FT10	10 × 10	930	930[a]	930	3.8	930	930	930	930
LA19	10 × 10	842	842[a]	842	0.5	842	842	842	842
LA21	15 × 10	1046	1046[a]	1046	15.2	1047	1046	1046	1046
LA24	15 × 10	935	935[a]	936.2	19.8	939	935	935	938
LA25	20 × 10	977	977[a]	977.1	13.8	977	977	977	979
LA27	20 × 10	1235	1235[a]	1235	11.7	1236	1235	1235	1235
LA29	20 × 10	1152	1153	1159.2	63.9	1160	1164	1157	1168
LA36	15 × 15	1268	1268[a]	1268	9.9	1268	1268	1268	1268
LA37	15 × 15	1397	1397[a]	1402.5	42.1	1407	1397	1397	1411
LA38	15 × 15	1196	1196[a]	1199.6	47.8	1196	1196	1196	1201
LA39	15 × 15	1233	1233[a]	1233.8	28.6	1233	1233	1233	1240
LA40	15 × 15	1222	1224	1224.5	52.1	1229	1224	1224	1233
ABZ7	20 × 15	656	658	661.8	85.9	670	664	662	666
ABZ8	20 × 15	665(645)	667	670.3	90.7	682	671	669	678
ABZ9	20 × 15	679(661)	678[b]	684.8	90.2	695	679	679	693
MRE			0.43	0 .67	–	1.04	0.61	0.53	1.09

[a]The best solutions found by our algorithm are equal to the best known lower bound.
[b]The best makespans our algorithm found are better than the best previously known value.

Meeran [11], the next columns named Best, M_{av}, T_{av} show the best makespan, average makesan and average computing time in seconds obtained by TSSA algorithm over 10 runs, respectively, and the last four columns show the results of the TSAB, SB-RGLS5, BV-best and TSSB. The last line shows the MRE in order to analyze the effectiveness of these algorithms.

Table 2 shows that the av-MRE provided by TSSA is lower than the MRE of TAAB and TSSB, and is close to the MRE of SB-RGLS5. The b-MRE provided by TSSA is better than that of BV-best. Moreover, TSSA can find the optimal solution for the notorious instance FT10 almost every time within four seconds on average. Even for the general 15 × 15 instances, for example LA36 instance, TSSA has the capability to find the optimal solution every time only in less than ten seconds on average. It must be pointed out that unlike the majority of algorithms which are initiated from the better initial solution generated by the specialized methods, TSSA algorithm obtains these results from randomly initial solutions. This indicates that TSSA algorithm is very robust and efficient. Among the fifteen instances, TSSA found the optimal solutions of ten (out of thirteen instances whose optimal solution values are known), and improved one upper bound among two unsolved instances, namely: ABZ9-D678.

Finally, the ORB class which contains 10 instances is analyzed. Table 3 lists the detailed results of comparison. In this table, TSSA is clearly better than TSAB, BV-best and TSSB in the solution quality. For example, TSSA found the optimal solutions for all instances, whereas BV-best, which is the best algorithm among the other three algorithms, found the optimal solutions for eight out of ten problems.

4.3. Results for instances (b)

YN class contains four instances with size 20 × 20, and no optimal solutions have been known. SWV class contains 20 instances with the number of operations between 200 and 500, and nine of them have not been solved for the optimal solutions. Instances SWV11-15 have been better solved by Zhang et al. [24] and instances SWV16-20 are relatively easy. Therefore, these two groups are omitted from the table. Benchmarks YN and SWV have not been tested by algorithm TSSB and algorithm *i*-TSAB did not provide the detailed results of each instance; therefore we primarily compare TSSA with SB-RGLS5 and BV-best.

Table 4 shows the detailed results of comparison for YN instances. TSSA algorithm outperforms the procedure BV both in the solution quality and computing time. For example, TSSA algorithm achieves av-MRE = 6.99% within 2

Table 3
Results for ORB01-10 instances

Problem	Size	LB	TSSA			TSAB′		BV-best[b]		TSSB[c]	
			Best	M_{av}	T_{av}(s)	Makespan	CI-CPU	Makespan	CPU(s)	Makespan	CPU(s)
ORB01	10 × 10	1059	1059[a]	1059	3.5	1059	548	1059	17.3	1064	82
ORB02	10 × 10	888	888[a]	888.1	6.4	890	376	888	88.4	890	75
ORB03	10 × 10	1005	1005[a]	1012.5	13.8	1005	356	1005	16.2	1013	87
ORB04	10 × 10	1005	1005[a]	1008.3	14.3	1011	427	1013	285.6	1013	75
ORB05	10 × 10	887	887[a]	888.6	6.6	889	389	889	15.2	887	81
ORB06	10 × 10	1010	1010[a]	1010	8.5	1013	472	1010	124.8	–	–
ORB07	10 × 10	397	397[a]	397	0.5	397	642	397	69.2	–	–
ORB08	10 × 10	899	899[a]	902.5	7.2	913	568	899	97.6	–	–
ORB09	10 × 10	934	934[a]	934	0.4	941	426	934	73.2	–	–
ORB10	10 × 10	944	944[a]	944	0.3	946	667	944	14.2	–	–
ORB01-10			0.0	0.17	6.2	0.37	487.1	0.10	80.2	0.46	80

[a]The best solutions found by our algorithm are equal to the best known lower bound.
[b]The CPU time on the SUN Sparc-330.
[c]The CPU time on personal computer Pentium 133 MHz.

Table 4
Results for YN1-4 instances

Problem	Size	UB(LB)	TSSA			BV[b]			
			Best	M_{av}	T_{av}(s)	SB-RGLS5	CPU(s)	BV-best	CPU(s)
YN1	20 × 20	885(826)	884[a]	891.3	106.3	893	3959.2	891	9382.4
YN2	20 × 20	909(861)	907[a]	911.2	110.4	911	5143.2	910	11647.2
YN3	20 × 20	892(827)	892	895.5	110.8	897	4016	897	4016
YN4	20 × 20	968(918)	969	972.6	108.7	977	7407.2	972	10601.2
YN1-4			6.4	6.99	109.1	7.2	5131.4	6.98	8911.7

[a]The best makespans our algorithm found are better than the best previously known value.
[b]The CPU time on the SUN Sparc-330.

min on our personal PC, whereas BV-best needs approximately 150 min on the SUN SParc-330 to achieve the similar aim. Moreover, TSSA algorithm found two new upper bounds, namely: YN1-884 and YN2-907.

Similarly as for YN instances, the detailed results for SWV instances are shown in Table 5 . In order to find a better solution, *TotIter* is set to $n \times 100\,000$ for SWV instances. The solution quality provided by TSSA algorithm outperforms that of BV in a short time. For example, for SWV06-10 instances, TSSA algorithm offers b-MRE = 6.91% in about 3 min on the personal PC, whereas BV-best provides b-MRE = 8.11% in approximately 180 min on the SUN SParc-330. Moreover, for SWV01-10 instances, TSSA algorithm found three new upper bounds, namely: SWV04-1470, SWV08-1756 and SWV10-1754.

4.4. Results for instances (c)

TA class contains 80 instances with the number of operations between 225 and 2000. Due to the increasing of the ratio of the number of jobs (n) to the number of machines (m), instances TA51-80 are relatively easy except for TA62 and TA67, and instances TA01-50 are particularly hard. Among the hard instances TA01-50, the optimal solutions are known only for 17 out of the 50 instances. The instances TA21-30 (20×20) and TA41-50 (30×20) are widely regarded as the most difficult JSP benchmark problems, and no one optimal solution has been found. Therefore, we focus our attention on instances TA01-50, and for instances TA62 and TA67 we only provide the results obtained. Because instances

Table 5
Results for SWV01-10 instances

Problem	Size	UB(LB)	TSSA			BV[c]			
			Best	M_{av}	T_{av}(s)	SB-RGLS5	CPU(s)	BV-best	CPU(s)
SWV01	20 × 10	1407	1412	1423.7	142.1	1418	1498	1418	1498
SWV02	20 × 10	1475	1475[a]	1480.3	119.7	1484	1389.2	1484	1389.2
SWV03	20 × 10	1398(1369)	1398	1417.5	139.1	1443	–	1425	3302
SWV04	20 × 10	1474(1450)	1470[b]	1483.7	143.9	1484	1621.2	1483	2433.2
SWV05	20 × 10	1424	1425	1443.8	146.7	1434	1961.2	1434	1961.2
SWV01-05			0.78	1.76	138.3	1.97	1617.2	1.69	2116.7
SWV06	20 × 15	1678(1591)	1679	1700.1	192.5	1710	5446	1696	11863
SWV07	20 × 15	1600(1446)	1603	1631.3	190.2	1645	3903.2	1622	10699
SWV08	20 × 15	1763(1640)	1756[b]	1786.9	190	1787	4264	1785	10375
SWV09	20 × 15	1661(1604)	1661	1689.2	193.8	1703	4855.2	1672	12151
SWV10	20 × 15	1767(1631)	1754[b]	1783.7	184.6	1794	3005.2	1773	10332
SWV06-10			6.91	8.66	190.2	9.27	4294.7	8.11	11084

[a]The best solutions found by our algorithm are equal to the best known lower bound.
[b]The best makespans our algorithm found are better than the best previously known value.
[c]The CPU time on the SUN Sparc-330

TA01-50 are particularly difficult, we are primarily interested in the solution performance not seen the running time. Therefore, *TotIter* is set to $n \times m \times 10000$ to find a better solution. For particularly hard instances TA41-50, the results are selected among two standard runs which apply the neighborhood structure N6 and neighborhood structure NS6 proposed by Zhang CY et al. [24], respectively. (Neighborhood structure NS6 can obtain the better quality of solution at the cost of increasing the running time.)

Tables 6 and 7 list the detailed comparison of performance of TSSA with those of SB-RGLS5, BV-best and TSSB. For TA01-50 instances, TSSA offers b-MRE = 2.49% and av-MRE = 2.95%, whereas SB-RGLS5, BV-best and TSSB provide 3.29%, 3.01% and 3.68%, respectively. TSSA clearly outperforms SB-RGLS5, BV-best and TSSB in the solution quality. Moreover, TSSA need the relatively short computing times to achieve these results on our personal computer. Next, we compare the performance of TSSA with that of *i*-TSAB, which is the current state-of-art approximation algorithm for the JSP. For TA01-50 instances, *i*-TSAB algorithm offers MRE = 2.82% [32]. In terms of av-MRE, TSSA only slightly underperforms *i*-TSAB with the small difference; In terms of b-MRE, TSSA outperforms *i*-TSAB with the large difference of 0.33. Without a well-defined "mean" performance metric being established, there is no access to assess the significance of the observed differences. However, according to rough comparison of the average MRE of TSSA with MRE of *i*-TSAB, TSSA might slightly underperform the intricate *i*-TSAB for the TA01-50 instances. Moreover, TSSA algorithm found the 11 new upper bounds among the 33 unsolved instances, see Table 6.

5. Conclusions

In this paper, a fast and easily implementable heuristics algorithm is presented for the job shop scheduling problem with makespan criterion. By the suitable use of the structure of the solution space (especially the big valley), the new hybrid TSSA algorithm is proposed. This algorithm appropriately combines the advantage properties of SA and TS, mitigating the drawback of TS. TSSA algorithm could reduce the influence of the initial solution and obtain the high-quality solutions in a short running time on a modern PC. These properties have been confirmed by tests on a large number of benchmark problems. For example, it improves a lot of the current best solutions within reasonable computing times. These indicate that TSSA algorithm is a very robust and efficient algorithm for the considered problem. The general idea of the hybrid of TS and SA could also be applied to solving the other difficult combinatorial optimization problems.

Table 6
Results for the TA01-50 instances

Problem	Size	UB(LB)	TSSA			BV		TSSB
			Best	M_{av}	T_{av}(s)	SB-RGLS5	BV-best	Makespan
TA01	15 × 15	1231	1231[a]	1231	11.2	1231	1231	1241
TA02	15 × 15	1244	1244[a]	1244.1	30.1	1250	1244	1244
TA03	15 × 15	1218	1218[a]	1219.4	108.5	1220	1218	1222
TA04	15 × 15	1175	1175[a]	1176.2	71.7	1181	1181	1175
TA05	15 × 15	1224	1224[a]	1224	10.8	1233	1233	1229
TA06	15 × 15	1238	1238[a]	1240.8	125.2	1245	1241	1245
TA07	15 × 15	1227	1228	1228	138.6	1228	1228	1228
TA08	15 × 15	1217	1217[a]	1217.1	27.6	1217	1217	1220
TA09	15 × 15	1274	1274[a]	1275.2	61.3	1274	1274	1291
TA10	15 × 15	1241	1241[a]	1246.6	68	1241	1241	1250
TA11	20 × 15	1361(1323)	1359[b]	1367.6	260.4	1388	1364	1371
TA12	20 × 15	1367(1351)	1371	1374.3	253.4	1377	1367	1379
TA13	20 × 15	1342(1282)	1342	1355.2	258.6	1350	1350	1362
TA14	20 × 15	1345	1345[a]	1346.7	65.2	1355	1345	1345
TA15	20 × 15	1340(1304)	1339[b]	1348.4	252.7	1353	1353	1360
TA16	20 × 15	1360(1302)	1360	1366.2	253.2	1372	1369	1370
TA17	20 × 15	1462	1464	1472.9	249.4	1479	1478	1481
TA18	20 × 15	1396(1369)	1399	1408.7	231.9	1396	1396	1426
TA19	20 × 15	1335(1297)	1335	1340.6	261.5	1348	1341	1351
TA20	20 × 15	1351(1318)	1350[b]	1356.7	264	1364	1359	1366
TA21	20 × 20	1644(1539)	1644	1650.5	437	1659	1659	1659
TA22	20 × 20	1600(1511)	1600	1606.4	433.5	1619	1603	1623
TA23	20 × 20	1557(1472)	1560	1564.5	429.4	1565	1558	1573
TA24	20 × 20	1647(1602)	1646[b]	1653.2	431.6	1660	1659	1659
TA25	20 × 20	1595(1504)	1597	1607.7	421	1615	1615	1606
TA26	20 × 20	1645(1539)	1647	1654.9	436.2	1668	1659	1666
TA27	20 × 20	1680(1616)	1680	1688.7	447.8	1689	1689	1697
TA28	20 × 20	1614(1591)	1603[b]	1616.6	431.2	1615	1615	1622
TA29	20 × 20	1625(1514)	1627	1630.1	426.2	1629	1629	1635
TA30	20 × 20	1584(1473)	1584	1597.7	436.1	1612	1604	1614
TA31	30 × 15	1764	1764[a]	1765.8	318.3	1766	1766	1771
TA32	30 × 15	1796(1774)	1795[b]	1811.7	613.2	1803	1803	1840
TA33	30 × 15	1793(1778)	1796	1806.7	494.7	1808	1796	1833
TA34	30 × 15	1829(1828)	1831	1831.8	337.9	1840	1832	1846
TA35	30 × 15	2007	2007[a]	2011	129.5	2007	2007	2007
TA36	30 × 15	1819	1819[a]	1820.5	178.1	1831	1823	1825
TA37	30 × 15	1778(1771)	1778	1784.4	496.4	1795	1784	1813
TA38	30 × 15	1673	1673[a]	1678.5	333	1681	1681	1697
TA39	30 × 15	1795	1795[a]	1806.6	303.2	1798	1798	1815
TA40	30 × 15	1674(1631)	1676	1684	499.3	1698	1686	1725
TA41	30 × 20	2018(1859)	2018	2028.5	805.5	2026	2026	2045
TA42	30 × 20	1956(1867)	1953[b]	1964.5	805.2	1974	1967	1979
TA43	30 × 20	1859(1809)	1858[b]	1882.6	922.8	1881	1881	1898
TA44	30 × 20	1984(1927)	1983[b]	1998.2	897.9	2016	2004	2036
TA45	30 × 20	2000(1997)	2000	2006.8	805.1	2018	2008	2021
TA46	30 × 20	2021(1940)	2010[b]	2029.2	818.8	2051	2040	2047
TA47	30 × 20	1903(1789)	1903	1918.2	877.4	1921	1921	1938
TA48	30 × 20	1952(1912)	1955	1968.6	802.6	1982	1982	1996
TA49	30 × 20	1968(1915)	1967[b]	1980	910.9	2008	1994	2013
TA50	30 × 20	1926(1807)	1931	1945.6	811.8	1967	1967	1975
TA62	50 × 20	2869	2869[a]	–	–	2900	2900	2942
TA67	50 × 20	2825	2825[a]	–	–	2826	2826	2865

[a]The best solutions found by our algorithm are equal to the best known lower bound.
[b]The best makespans our algorithm found are better than the best previously known value.

Table 7
Comparison with the other three algorithms for TA01-50 instances

Problem group	TSSA			SB-RGLS5[a]		BV-best[a]		TSSB[b]	
	b-MRE	av-MRE	T_{av}(s)	MRE	T_{av}(s)	b-MRE	T_{av}(s)	MRE	T_{av}(s)
TA01-10	0.01	0.11	65.3	0.25	1057.3	0.16	1498	0.45	2175
TA11-20	2.37	2.92	235	3.26	2771.1	2.81	4559	3.47	2526
TA21-30	5.43	5.97	433	6.37	4475.2	6.10	6850	6.52	34910
TA31-40	0.55	0.93	370.4	1.08	5231.7	0.80	8491	1.92	14133
TA41-50	4.07	4.84	845.8	5.48	13107.3	5.20	16018	6.04	11512
TA01-50	2.49	2.95	389.9	3.29	5328.5	3.01	7483	3.68	13051

[a]The CPU time on the SUN Sparc-330.
[b]The CPU time on personal computer Pentium 133 MHz.

Acknowledgments

This paper is supported by the National Basic Research Program of China (under the Grant No. 2005CB724107) and the National Natural Science Foundation, China (under the Grants no. 70271053, 50305008). The authors would like to thank the referees for their helpful comments and suggestions.

References

[1] Garey EL, Johnson DS, Sethi R. The complexity of flowshop and job-shop scheduling. Mathematics of Operations Research 1976;1:117–29.
[2] Muth JF, Thompson GL. Industrial scheduling. Englewood Cliffs, NJ: Prentice-Hall; 1963.
[3] Adams J, Balas E, Zawack D. The shifting bottleneck procedure for job shop scheduling. Management Science 1988;34:391–401.
[4] Croce FD, Tadei R, Volta G. A genetic algorithm for the job shop problem. Computers & Operations Research 1995;22:15–24.
[5] Dorndorf U, Pesch E. Evolution based learning in a job shop scheduling environment. Computers & Operations Research 1995;22:25–40.
[6] Van Laarhoven PJM, Aarts EHL, Lenstra JK. Job shop scheduling by simulated annealing. Operations Research 1992;40(1):113–25.
[7] Steinhöfel K, Albrecht A, Wong CK. Two simulated annealing-based heuristics for the job shop scheduling problem. European Journal of Operational Research 1999;118(3):524–48.
[8] Taillard ÉD. Parallel taboo search techniques for the job-shop scheduling problem. ORSA Journal on Computing 1994;6:108–17.
[9] Nowicki E, Smutnicki C. A fast taboo search algorithm for the job shop scheduling problem. Management Science 1996;42(6):797–813.
[10] Blażewicz J, Domschke W, Pesch E. The job shop scheduling problem: Conventional and new solution techniques. European Journal of Operational Research 1996;93:1–33.
[11] Jain AS, Meeran S. Deterministic job shop scheduling: past, present and future. European Journal of Operational Research 1999;113:390–434.
[12] Glover F. Future paths for integer programming and links to artificial intelligence. Computers & Operations Research 1986;13:533–49.
[13] Kirkpatrick S, Gelatt CD, Vecchi MP. Optimization by simulated annealing. Science 1983;220:671–90.
[14] Glover F. Tabu search–Part I. ORSA Journal on Computing 1989;1(3):190–206.
[15] Glover F. Tabu search–Part II. ORSA Journal on Computing 1990;2(1):4–32.
[16] Glover F, Laguna M. Tabu search. Dordrecht: Kluwer Academic Publishers; 1997.
[17] Taillard ÉD. Parallel taboo search technique for the jobshop scheduling problem. Technical report ORWP 89/11, DMA, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 1989.
[18] Dell'Amico M, Trubian M. Applying tabu-search to job-shop scheduling problem. Annals of Operations Research 1993;41(1–4):231–52.
[19] Barnes JW, Chambers JB. Solving the job shop scheduling problem using tabu search. IIE Transactions 1995;27:257–63.
[20] Chambers JB, Barnes JW. New tabu search results for the job shop scheduling problem. Technical report ORP96-10, Graduate Program in Operations Research and Industrial Engineering, The University of Texas at Austin, 1996.
[21] Jain AS, Rangaswamy B, Meeran S. New and "stronger" job-shop neighbourhoods: A focus on the method of Nowicki and Smutnicki (1996). Journal of Heuristics 2000;6(4):457–80.
[22] Nowicki E, Smutnicki C. Some new ideas in TS for job-shop scheduling. Technical report 50/2001, Institute of Engineering Cybernetics, Wroclaw University of Technology, Wroclaw, Poland, 2001.
[23] Glover F, Hanafi S. Tabu search and finite convergence. Discrete Applied Mathematics 2002;119(1–2):3–36.
[24] Zhang CY, Li PG ,Guan ZL, Rao YQ. A Tabu Search Algorithm with a New Neighborhood Structure for the Job Shop Scheduling Problem. Computers & Operations Research, in press, doi: 10.1016/j.cor.2005.12.002 .
[25] Balas E, Vazacopoulos A. Guided local search with shifting bottleneck for job shop scheduling. Management Science 1988;44(2):262–75.
[26] Lawrence S. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement). Graduate School of Industrial Administration. Pittsburgh, Pennsylvania: Carnegie-Mellon University; 1984.
[27] Applegate D, Cook W. A computational study of job-shop scheduling. ORSA Journal on Computer 1991;3(2):149–56.
[28] Yamada T, Nakano R. A genetic algorithm applicable to large-scale job-shop problems. In: Manner R, Manderick B, editors. Proceedings of the Second international workshop on parallel problem solving from Nature (PPSN'2). Belgium: Brussels; 1992. p. 281–90.

[29] Storer RH, Wu SD, Vaccari R. New search spaces for sequencing problems with applications to job-shop scheduling. Management Science 1992;38(10):1495–509.
[30] Taillard ÉD. Benchmarks for basic scheduling problems. European Journal of Operational Research 1993;64(2):278–85.
[31] Taillard ÉD. http://ina.eivd.ch/Collaborateurs/etd/default.htm. January, 2005.
[32] Nowicki E, Smutnicki C. Some new tools to solve the job shop problem. Technical report 60/2002, Institute of Engineering Cybernetics, Technical University of Wroclaw, Wroclaw, Poland, 2002.
[33] Pezzella F, Merelli E. A tabu search method guided by shifting bottleneck for job shop scheduling problem. European Journal of Operational Research 2000;120:297–310.