Some New Results on Tabu Search Algorithm Applied to the Job-Shop Scheduling Problem

Chapter · September 2008

DOI: 10.5772/5593 · Source: InTech

Chaoyong Zhang
Huazhong University of Science and Technology
159 PUBLICATIONS **2,229** CITATIONS
SEE PROFILE

Xinyu Shao
Soochow University (PRC)
178 PUBLICATIONS **2,809** CITATIONS
SEE PROFILE

Haobo Qiu
Huazhong University of Science and Technology
79 PUBLICATIONS **716** CITATIONS
SEE PROFILE

Some New Results on Tabu Search Algorithm Applied to the Job-Shop Scheduling Problem

Chaoyong Zhang, Xinyu Shao, Yunqing Rao and Haobo Qiu
Key Laboratory of Digital Manufacturing Equipment & Technology, School of Mechanical Science and Engineering, Huazhong University of Science and Technology, Wuhan 430074 China

1. Introduction

The Job-Shop Scheduling Problem (JSSP) with the makespan criterion comes from the manufacturing industry and has excellent practical applications. The problem can be briefly described as follows. There are a set of jobs and a set of machines. Each job consists of a sequence of operations, and each of the operations uses one of the machines for a fixed duration. Each machine processes at most one operation at one time. Once the operation started, no preemption is permitted. A scheduling is an assignment of operations to time intervals on the machines. The objective of the problem is to find a schedule which minimizes the makespan (C_{max}), that is, the finish time of the last operation completed.
The JSSP is widely acknowledged as one of the most difficult NP-complete problems (Garey et al., 1976). This is illustrated by the fact that a relatively small instance with 10 jobs and 10 machines, proposed by Muth & Thompson (1963), remained unsolved for more than a quarter of a century, and until now no problems are solved to optimality for the 20×20 instances. Since it is an important practical problem, the JSSP has captured the interest of a significant number of researchers during the past three decades, and many optimization algorithms and approximation algorithms have been proposed. The optimization algorithms, which are primarily based on the B&B scheme (Carlier & Pinson, 1989; Brucker et al., 1994), have been successfully applied to solving small instances. However, they cannot accomplish optimal schedules in a reasonable time for instances larger than 250 operations with reached the limit. On the other hand, approximation algorithms, which include priority dispatch, shifting bottleneck approach, meta-heuristic methods and so on, provide a quite good alternative for the JSSP. Approximation algorithms were firstly developed on the basis of dispatching rules (Giffler & Thompson, 1960), which are very fast, but the quality of solutions that they provide usually leaves plenty of room for improvement. A more elaborate algorithm, which could produce considerably better approximations at a higher computational cost, is the shifting bottleneck approach proposed by Adams et al. (Adams et al., 1988). More recently, the meta-heuristic methods, such as tabu search (TS) (Taillard, 1994; Nowicki & Smutnicki, 1996), simulated annealing (SA) (Van Laarhoven et al., 1992), genetic algorithm (GA) (Croce et al., 1995), could provide the good solutions for a large scale problem and have captured the attention of many researchers. Moreover, most recent studies indicate that a single technique cannot solve this stubborn

Source: Local Search Techniques: Focus on Tabu Search, Book edited by: Wassim Jaziri, ISBN 978-3-902613-34-9, pp. 278, October 2008, I-Tech, Vienna, Austria

problem. Much work has been directed on hybrid methods involving GA, TS, SA and SB techniques, as hybrid methods are able to provide high-quality solutions within reasonable computing times. The relevant surveys can be seen form Vaessens et al. (1996), Błażewicz et al. (1996) and Jain & Meeran (1999).

Within the class of meta-heuristic methods, Tabu search, initially proposed by Glover (Glover, 1989 1990; Glover & Laguna, 1997) and Hansen (Hansen, 1986), currently seems to be one of the most promising methods for the job shop scheduling problem with the makespan criterion. It uses a memory function to avoid being trapped at a local minimum. Tabu search was firstly applied to the JSSP by Taillard (1989), whose main contribution was the use of the neighborhood structure introduced by Van Laarhoven et al. (1992) and the presentation of a fast move estimation strategy. Furthermore, Taillard (1989) observed that this algorithm has a higher efficiency for rectangular instances. Since then, researchers have introduced numerous improvements to Taillard's original algorithm, and the most important contributions are made by a myriad of researchers among whom are Nowicki & Smutnicki (1996), Dell'Amico & Trubian (1993), Barnes & Chambers (1995) and Chambers & Barnes (1996). Among these individual tabu search methods, algorithm TSAB designed by Nowicki & Smutnicki (1996) introduces the real breakthrough in both efficiency and effectiveness for the JSSP. For example, it finds the optimal solution for the notorious instance FT10 within only 30 seconds on a now-dated personal computer. The *i*-TSAB technique of Nowicki & Smutnicki (2002), which is an extension of their earlier TSAB algorithm, represents the current state-of-the-art approximation algorithm for the JSSP and improves the majority of upper bounds of the unsolved instances.

Although tabu search has emerged as an effective algorithmic approach for the JSSP, it was initially designed to find the near-optimum solution of combinatorial optimization problems and no clean proof of convergence is known (Hanafi, 2000). Like many local searches the quality of the best solution found by tabu search approach depends on the initial solution and neighborhood structures. In this paper, two innovative approaches are proposed to overcome these problems for the JSSP. Firstly, a new neighborhood structure is proposed to solve the job shop scheduling problem by tabu search approach. Secondly, by reasonably combining the memory function (avoid cycling) of tabu search and the convergent characteristics of simulated annealing, we develop an efficient hybrid optimization algorithm. In this approach, simulated annealing is used to find the sufficient "good" solutions over the big valley so that tabu search can re-intensify searches from the promising solutions.

In the end, this algorithm is tested on the commonly standard benchmark set and compared with the other approaches. The computational results show that the proposed algorithm could reduce the influence of the initial solution and obtain the high-quality solutions within reasonable computing times. These have been confirmed by tests on a large number of benchmark problems. For example, some new upper bounds among the unsolved problems are found in a short time.

The remainder of this paper is organized as follows. Section 2 gives the representation of the job shop scheduling problem. In Section 3, the framework of the hybrid of tabu search and simulated annealing is provided. Section 4 presents the implementation of TSSA algorithm for the JSSP and the proposed neighborhood structure. In Section 5, we firstly present the comparison of the different neighborhood structures and comparison of move evaluation strategies respectively, and then give the computational and comparative results on the benchmark instances. Conclusion is presented in Section 6.

2. Representation of the JSSP

The job shop scheduling problem can be represented with a disjunctive graph (Balas, 1969). Let $J = \{1, 2, \ldots, n\}$ be the set of jobs, $M = \{1, 2, \ldots, m\}$ the set of machines. A disjunctive graph $G := (V, A, E)$ is defined as follows: V is $\{0, 1, 2, \ldots, \tilde{n}\}$ the set of nodes representing all operations where 0 and $\tilde{n}$ represent the dummy start and finish operations, respectively. A is the set of conjunctive (directed) arcs connecting consecutive operations of the same job, and E is the set of disjunctive arcs connecting operations to be processed by the same machine k. More precisely, $E = \bigcup_{k=1}^{m} E_k$, where E_k is the subset of disjunctive pair-arcs corresponding to machine k; each disjunctive arc of E can be considered as a pair of oppositely directed arc. The length of an arc $(i, j) \in A$ is p_i that denotes the processing time. The length of each arc $(i, j) \in E$ is either p_i or p_j depending on its orientation. Let us consider an example of the three jobs and three machines given in Table 1. This problem can be represented by a disjunctive graph shown in Fig. 1.

Job	(Machine sequence, Processing time)		
J1	(1, 3)	(2, 2)	(3, 5)
J2	(1, 3)	(3, 5)	(2, 1)
J3	(2, 2)	(1, 5)	(3, 3)

Table 1. An example of three jobs and three machines

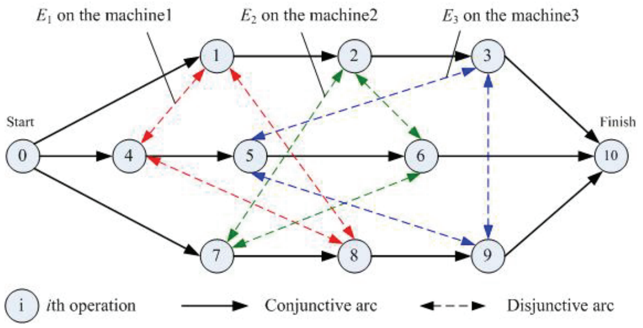

Fig. 1. The disjunctive graph of an instance with $n = 3$, $m = 3$, and $\tilde{n} = 10$

According to the Adams et al. (1988) method, the graph G can be decomposed into the direct sub-graph $D = (V, A)$, by removing disjunctive arcs, and into m cliques $G_k = (V_k, E_k)$, obtained from G by deleting both the conjunctive arcs and the dummy nodes 0 and $\tilde{n}$. A selection S_k in E_k contains exactly one directed arc between each pair of oppositely directed arcs in E_k. A selection is acyclic if it does not contain any directed cycle. Moreover, sequencing machine k means choosing an acyclic selection in E_k. A complete selection S consists of the union of selections S_k, one of each E_k, $k \in M$. A complete selection S, i.e., replacing the disjunctive arc set E with the conjunctive arc set S, gives rise to directed graph $D_s = (V, A \cup S)$; A complete selection S is acyclic if the digraph D_s is acyclic. An acyclic selection S defines a schedule, i.e., a feasible solution of problem. Fig. 2 represents a feasible solution for the disjunctive graph in Fig. 1. Furthermore, if $L(u, v)$ denotes the length of a longest path from u to v in D_s, then the makespan $L(0, \tilde{n})$ of the schedule is equal to the length of a longest path in D_s. Therefore, in the language of disjunctive graphs, to solve the job shop scheduling problem is to find an acyclic complete selection $S \subset E$ that minimizes the length of the longest (critical) path in the directed graph D_s.

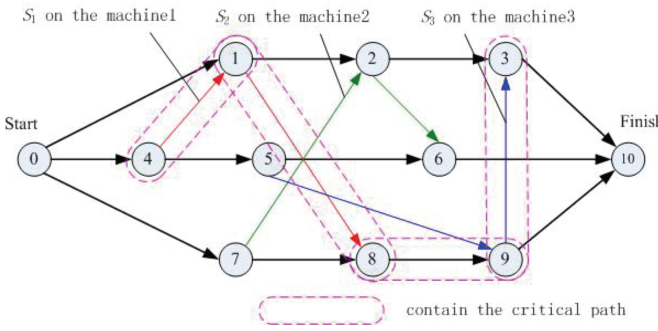

Fig. 2. A feasible solution for the disjunctive graph in Fig. 1

A key component of a feasible solution is the critical path, which is the longest route from start to finish in directed graph D_s= (V, $A \cup S$) and whose length represents the makespan C_{max}. Any operation on the critical path is called a critical operation. In Fig. 2 the length of the critical path is 19 and the critical path is {0, 4, 1, 8, 9, 3, 10}. It is also possible to decompose the critical path into a number of blocks. A block is a maximal sequence of adjacent critical operations that is processed on the same machine. In Fig. 2 the critical path is divided into two blocks, B_1 = {4, 1, 8} and B_2 = {9, 3}. Any operation, u, has two immediate predecessors and successors, with its job predecessor and successor denoted by $JP[u]$ and $JS[u]$ and its machine predecessor and successor denoted by $MP[u]$ and $MS[u]$. In other words, ($JP[u]$, u) and (u, $JS[u]$) are arcs of the conjunctive graph D_s, ($MP[u]$, u) and (u, $MS[u]$) (if they exist) are arcs of S.

In the JSSP, small perturbations are generally produced by re-ordering the sequence of operations on a critical path, and only through such re-ordering is it possible to produce a neighbor with a makespan better than that of the current solution.

3. The neighborhood structure

A neighborhood structure is a mechanism which can obtain a new set of neighbor solutions by applying a small perturbation to a given solution. Each neighbor solution is reached immediately from a given solution by a move (Glover & Laguna, 1997). Neighborhood structure is directly effective on the efficiency of tabu search algorithm. Therefore, unnecessary and infeasible moves must be eliminated if it is possible.

The most general neighborhood definition consists of swapping any adjacent pair of operations on the same machine. This neighborhood is quite large, and requires considerable effort to identify and evaluate the schedule that results from each possible move. For large problems the neighborhood contains more moves than can be examined and evaluated within a reasonable time. In addition, some of the moves can result in non feasible schedules. Consequently work has been devoted to the goal of reducing the size of this neighborhood and guaranteeing feasibility but without affecting solution quality (Jain et al., 2000).

The first successful neighborhood structure for the JSSP was introduced by van Laarhoven et al. (1992), and is often denoted by N1, see Fig. 3 (N1 is first named by Blazewicz et al (1996), and N2, N4, N5 and N6 are named in the same way). Their neighborhood structure, derived from the seminal work of Balas (1969), has laid the foundations for the most effective search strategies currently employed. The N1 neighborhood is generated by swapping any adjacent pair of critical operations on the same machine, and based on the following properties:

- Given a feasible solution, the exchange of two adjacent critical operations cannot yield an infeasible solution;
- The permutation of non-critical operations cannot improve the objective function and even may create an infeasible solution;
- Starting with any feasible solution, there exists some sequence of moves that will reach an optimal solution (known as the connectivity property)[1].

However, the size of the neighborhood N1 is quite large and includes a great number of unimproved moves. In order to reduce the number of block moves, Matsuo *et al*. (1988) proved that **unless the job-predecessor of u or the job-successor of v is on the critical path $P(0,n)$, the interchange containing u and v cannot reduce the makespan,** i.e. swapping internal operations within a block never gives an immediate improvement on the makespan. The work of Matsuo et al. (1988) allowed the neighborhood of moves to be reduced quite substantially.

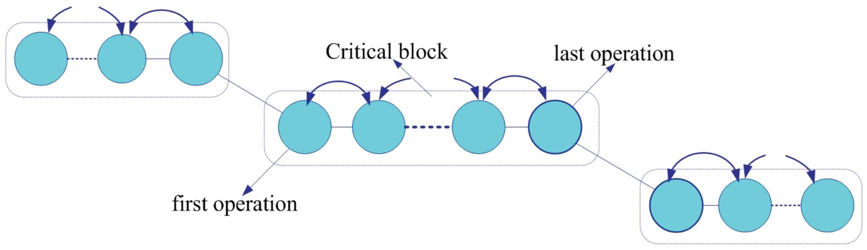

Fig. 3. The N1 neighborhood of moves

A neighborhood due to Grabowski *et al*. (1988), based on extending a neighborhood for a one machine problem (Grabowski *et al*. 1986), provides the next advance. This work introduced the concept of a block and defined a move to consist of inserting an operation at either the front or the rear of the critical block. Then, further refinements have been provided by Dell'Amico & Turbian(1993) (N4), Nowicki & Smutnicki (1996) (N5) and Balas & Vazacopoulos (1998) (N6).

The neighborhood N4 moves all operations *i* in a block to the very beginning or to the very end of this block, (Dell'Amico and Turbian proposed two neighborhood structures N3 and N4; in this paper we only discuss the neighborhood N4), N4 neighborhood structure is connected. The neighborhood N5 involves the reversal of a single border arc of a critical

1 However Kolonko (1998) proves that the connectivity property does not imply convergence to an optimum in these neighbourhoods.

block[2] and is substantially smaller (or more constrained) than the other neighborhoods, whereas the neighborhood N4 and N6 involves the reversal of more than one disjunctive arc at a time and thus could investigate a considerably larger neighborhood. The neighborhood N6, which is also considered as an extension of the neighborhood N5, is more constrained (smaller) than the neighborhood N4 and is currently one of the most efficient neighborhood structures. These neighborhoods N4, N5 and N6 are illustrated in Fig. 4, Fig. 5 and Fig. 6, respectively.

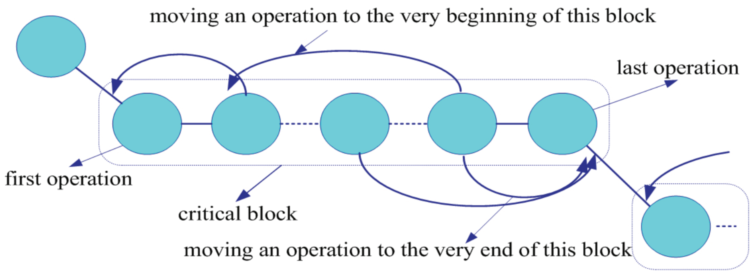
moving an operation to the very beginning of this block
last operation
first operation
critical block
moving an operation to the very end of this block

Fig. 4. The N4 neighborhood of moves

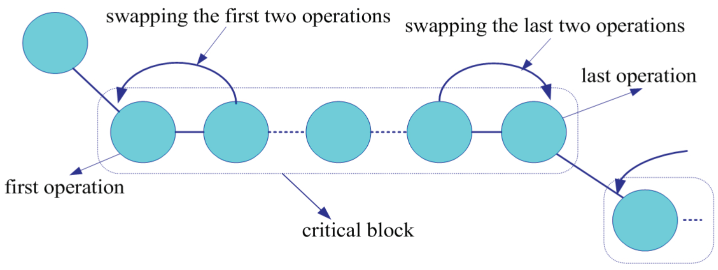
swapping the first two operations
swapping the last two operations
last operation
first operation
critical block

Fig. 5. The N5 neighborhood of moves

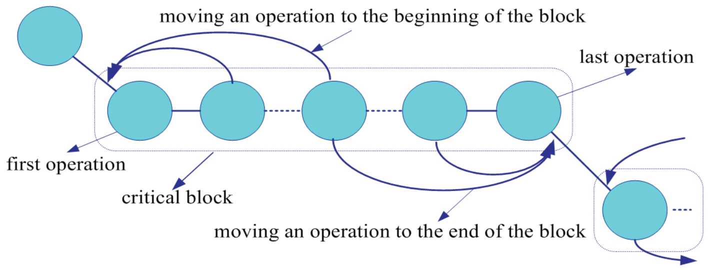
moving an operation to the beginning of the block
last operation
first operation
critical block
moving an operation to the end of the block

Fig. 6. The N6 neighborhood of moves

[2] In this neighborhood only one critical path is generated. A move is defined by the interchange of two successive operations *i* and *j*, where *i* or *j* is the first or last operation in a block that belongs to a critical path. In the first block only the last two operations and symmetrically in the last block of the critical path only the first two operations are swapped.

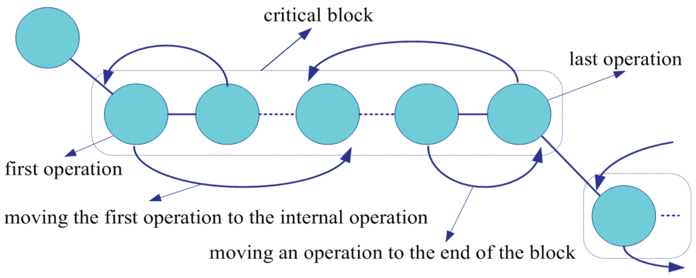
critical block
last operation
first operation
moving the first operation to the internal operation
moving an operation to the end of the block

Fig. 7. The further extended neighborhood structure

According to Matsuo et al.(1988), we have seen that in order to achieve an improvement by an interchange on u and v (assume u is processed before v), either $JP[u]$ or $JS[v]$ must be contained on the critical path P $(0, n)$, that is, either u or v must be the first or last operation of a critical block. Therefore, a further extended neighborhood used in this paper is proposed, **which not only inserts an operation to the beginning or the end of the critical block, but also moves the first or the last operation into the internal operation within the block,** illustrated in Fig. 7. This leads to a considerably larger neighborhood and investigates a much larger space. However, the questions to be explored are under what conditions an interchange on critical operation u and v is guaranteed not to create a cycle and how to reduce the neighborhood size. Next we give the two theorems and the proof for our neighborhood structure. Consider a feasible solution s:

Theorem 1.

If two operations u and v to be performed on the same machine are both on the critical path P $(0, n)$, and $L(v, n) \geq L(JS[u], n)$, then moving u right after v yields an acyclic complete selection.

Proof. By contradiction: suppose moving u right after v create a cycle C. Then C contains either $(u, JS[u])$ or $(u, MS[v])$. If $(u, JS[u]) \in C$, there is a path from $JS[u]$ to v in D_s(the cycle is $JS[u] \rightarrow v \rightarrow u \rightarrow JS[u]$), and hence $L(JS[u], n) > L(v, n)$, contrary to assumption. If $(u, MS[v]) \in C$, there is a path from $MS[v]$ to v in D_s, contrary to the assumption that D_s is acyclic. This completes the proof.

Theorem1 derives the idea that moving a critical operation u right after a critical operation v will not create a cycle if there is no directed path from $JS[u]$ to v in D_s. The Theorem1 could also be described briefly as follows: Given a feasible solution, if exchange two critical operation u and v, and the start time of the operation JS[u] is more than or equal to the start time of the operation v, then moving u right after v yields a feasible solution. We notice that if u and v are adjacent in critical path P $(0, n)$, then the conditions described are always satisfied (This is N1 neighborhood structure).

By analogy, we have Theorem 2.

Theorem 2.

If two operations u and v to be performed on the same machine are both on the critical path P $(0, n)$, and $L(0, u)+p_u \geq L(0, JP[v])+p_{JP[v]})$, then moving v right before u yields an acyclic complete selection.

Proof. Parallels that of theorem 1.

In order to construct our neighborhood structure, we in fact extend the scope of the Proposition2.2 and Proposition2.3 proposed by Balas & Vazacopoulos (1998) and present Theorem 1 and Theorem 2. Theorem 1 and Theorem 2 could be applied to an interchange on

any two *critical* operations u and v to be performed on the same machine, whether or not it contains either the $JS[u]$ or $JP[v]$ on the critical path. In our experiment, we observe that the neighborhood constructed by Theorem 1 and Theorem 2 is simpler and more constrained (smaller) than the similar neighborhood N4. Therefore, our search neighborhood could now be concisely defined as follows:
(1) If a critical path P (0, n) containing u and v also contains $JS[v]$, then insert u right after v and move v right before internal operations;
(2) if a critical path P (0, n) containing u and v also contains $JP[u]$, then insert v right before u and move u right after internal operations.
Assume that an interchange on two operation u and v results in a makespan increase of the new schedule compared to the old one. Then it is obvious that every critical path in new schedule contains the arc (v, u) (Balas & Vazacopoulos, 1998). We could make use of the fact and further reduce our neighborhood size.

4. The framework of the hybrid of TS and SA

Tabu search, defined and developed primarily by Glover (1989, 1990), has been successfully applied to a large number of combinatorial optimization problems, especially in production scheduling domain. Tabu search is an enhancement of the hill climbing heuristic. In order to avoid cycling through previous solutions, a short-term memory structure known as the tabu list is implemented. According to Brucker (Brucker, 1995), tabu search is an intelligent search that uses a memory function in order to avoid being trapped at a local minimum. Its goal is to emulate intelligent uses of memory, that is, tabu search tries to create memory itself similar to use of some memory functions of people in order to find its way out.
Tabu search algorithm was first applied to the JSSP by Taillard. Since then, researchers have introduced numerous improvements to Taillard's original algorithm. Among these tabu search methods, algorithm TSAB developed by Nowicki & Smutnicki (1996) introduces the significant breakthrough in both effectiveness and efficiency for the JSSP. 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 (Jain et al., 2000). By contrast, simulated annealing is not a powerful technique for the JSSP, but the initial solution has little influence on the solution quality obtained by simulated annealing procedure. However, due to lack of the memory function, simulated annealing may return to old solutions and become oscillation in local optimum surrounding. This causes the search to consume excessive computational times. 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. Therefore, in this paper, by exploiting the properties of the JSSP and the complementary strengths and weaknesses of the two paradigms, we present the newly designed hybrid TSSA algorithm.
The idea of the hybrid approach for the JSSP is based on the two important observations, due to Nowicki & Smutnicki (2001), as follows. First, the space structure of the considered job shop problem owns big valley (BV) and the best elite solutions dispersed over BV area. Second, tabu search is perfectly attracted to big valley area. Even though the initial solution was set relatively far from the valley, elite solutions generated by tabu search can still be collected inside big valley. The two observations indicate that tabu search is suitable for finding good solutions inside the big valley, and these good solutions previously

encountered provide better starting points for further space exploration than various initial solutions do. However, the number of solutions inside big valley is so large that it is unrealistic to expect that the whole valley might have been exhaustively searched. Nevertheless, simulated annealing, 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, which simulated annealing is used to find the promising elite solutions inside big valley generated on the search history and tabu search intensifies search around the solutions, is proposed. The main idea of TSSA algorithm is also related to the strategy that designs the more efficient forms of finite convergent tabu search based on recency-memory (Glover et al., 2002).The general framework of hybrid TSSA algorithm for the JSSP is outlined in Fig. 8.

```
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 $(f(s^*) < f(s^b) \mid\mid \exp(f(s^b) - f(s^*))/T > \text{random }[0, 1])$
            {
            "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 $iter \leq ImproveIter$ then go to Step 2.
Step 6. If a termination criterion is satisfied then stop. Else "pop" a solution from the
elite solution stack L, shift the solution to active schedule and store the active solution
as the current solution $s^*$, set $iter = 0$ and empty tabu list, then go to Step 2.
```

Fig. 8. Outline of TSSA algorithm for the JSSP

In the hybrid TSSA algorithm, the core tabu search is a straightforward implementation of tabu search intensification strategy. A strong diversification strategy using simulated annealing procedure to find the elite solutions inside big valley is equipped with the core tabu search and directs the intensified search to other regions of the solution space. More precisely, starting from a randomly initial solution, TSSA algorithm executes the core tabu search procedure and tracks the sufficiently "good" solutions found by simulated annealing procedure on the search history. The "good" solutions found by simulated annealing 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 the given value or the solution is proved to be optimal.

It can be seen that the hybrid TSSA framework in Fig. 8 can be converted to the traditional tabu search by omitting the simulated annealing unit, whereas it can be converted to a general simulated annealing by setting the length of tabu list to zero and the length of solution stack L to one. Such hybrid TS/SA strategy retains advantages of tabu search and simulated annealing, and provides a promising methodology to solve the other computationally intractable problems. For different problems, the neighborhood structure, parameters and algorithm criteria should be designed appropriately. Due to utilizing the properties of the JSSP, the new hybrid algorithm is closer resemblance to tabu search. According to the characteristics of the different problems, it even may develop the different TS/SA framework, such as the SATS strategy which most closely resembles simulated annealing and incorporates tabu search into simulated annealing. In the next section, a detailed description of each component function of TSSA algorithm for the JSSP is provided.

5. The implementation of TSSA

5.1 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 testing shows that the initial solution methods affect the solution quality for tabu search algorithm, so that the better initial solution might provide better results. Thus, for the majority of the tabu search, the specialized initialization procedure is used to obtain the better initial solution. However, the initial solutions have little influence on the solution quality provided by TSSA algorithm, but are effective on the running time. Hence the search is initiated from the randomized active solution.

5.2 The neighborhood structure

Neighborhood structure and move evaluation strategies are directly effective on the efficiency of the search for the JSSP, 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. In the TSSA algorithm, taking account on a balance of the effectiveness and efficiency, if the number of operations is less than 200, then N6 neighborhood structure introduced by Balas and Vazacopoulos (1998) is adopted; otherwise, the neighborhood structure proposed in this paper is applied.

5.3 Move evaluation

The run-time of local search algorithm for the JSSP is typically dominated by the cost of computing each move. Therefore, in order to make the algorithm more efficiently, a number of neighborhood evaluation strategies, such as exact methods and estimation methods, have been proposed. The procedure suggested by Ten Eikelder et al.(1999), called *bowtie*, is the most efficient exact method for the JSSP, which only recalculates the head and the tail values of the operations that need to be updated after the move. However, this exact method still takes a very long computational time for the larger instances. In order to accelerate the search process, the estimation methods which can quickly filter out moves that have a high probability of directing the search to new elite solution have been proposed. A fast estimation approach has been presented by Taillard (1994). A further accurate approach of this strategy has also been proposed by Nowicki and Smutnicki (2002), which is employed

in the famous *i*-TSAB algorithm. But these estimation strategies are only adapted to swap the adjacent operations of the critical block. A significant estimation strategy proposed by Balas and Vazacopoulos (1998) could be applied to reverse more than one disjunctive at a time, and the procedure of Balas and Vazacopoulos is also one of the most efficient implementations to solve the JSSP. The procedure of the estimation strategy proposed by Balas and Vazacopoulos is given as follows.
$L(i, j)$ and $L^{u,v}(i, j)$ denoted as the length of a longest path from i to j (if it exists) before and after an interchange on u and v, respectively, and by$\lambda^{u,v}(i, j)$ our evaluation (estimate) of $L^{u,v}(i, j)$, namely

$$\lambda^{u,v}(0, n) = \max\{\lambda^{u,v}(0, w)+\lambda^{u,v}(w, n):\ w\in Q\} \tag{1}$$

where $Q := \{u, l_1, \ldots,l_k, v\}$ is the segment of the critical path $P(0,n)$ containing u and v. Here the estimates$\lambda^{u,v}(0, w)$ and$\lambda^{u,v}(w, n)$ are calculated recursively as follows.
Case 1. The interchange on u and v is a forward one, then we have

$$\lambda^{u,v}(0,l_1)=\begin{cases} L(0,JP[l_1]+p_{JP[l_1]} & \text{if } u \text{ was the first operation on its machine,}\\ \max\{L(0,JP[l_1]+p_{JP[l_1]},L(0,MP[u])+p_{MP[u]}\} & \text{otherwise;}\end{cases}$$

for $w\in\{l_2, \ldots,l_k, v\}$,

$$\lambda^{u,v}(0,w)=\max\{L(0,JP[w]+p_{JP[w]},\lambda^{u,v}(0,MP[w])+p_{MP[w]}\},$$

and

$$\lambda^{u,v}(0,u)=\max\{L(0,JP[u]+p_{JP[u]},\lambda^{u,v}(0,v)+p_v\}$$

Further, we have

$$\lambda^{u,v}(u,n)=\begin{cases} p_u+L(JS[u],n) & \text{if } v \text{ was the last operation in its machine,}\\ p_u+\max\{L(JS[u],n),L(MS[v],n)\} & \text{otherwise,}\end{cases}$$

$$\lambda^{u,v}(v,n)=p_v+\max\{L(JS[v],n),\lambda^{u,v}(u,n)\},$$

and for $w\in\{l_1, \ldots,l_k\}$,

$$\lambda^{u,v}(w,n)=p_w+\max\{L(JS(w),n),\lambda^{u,v}(MS(w),n)\}.$$

Case 2. The interchange on u, v is a backward one. Then we have

$$\lambda^{u,v}(0,v)=\begin{cases} L(0,JP[v])+p_{JP[v]} & \text{if } u \text{ as the first operation in its machine,}\\ \max\{L(0,JP[v])+p_{JP[v]},L(0,MP[u])+p_{MP[u]}\} & \text{otherwise,}\end{cases}$$

$$\lambda^{u,v}(0,u)=\max\{L(0,JP[u])+p_{JP[u]},\lambda^{u,v}(0,v)+p_v\},$$

and for $w\in\{l_1, \ldots,l_k\}$,

$$\lambda^{u,v}(0,w)=\max\{L(0,JP[w])+p_{JP[w]},\lambda^{u,v}(0,MP[w])+p_{MP[w]}\}$$

Further, we have

$$\lambda^{u,v}(l_k,n)=\begin{cases} p_{l_k}+L(JS[l_k],n) & \text{if } u \text{ was the first operation on its machine,}\\ p_{l_k}+\max\{L(JS[l_k],n),L(MS[v],n)\} & \text{otherwise;}\end{cases}$$

for $w \in \{u, l_2, \ldots, l_{k-1}\}$,

$$\lambda^{u,v}(w,n)=p_w+\max\{L(JS[w],n),\lambda^{u,v}(MS[w],n)\},$$

And

$$\lambda^{u,v}(v,n)=p_v+\max\{L(JS[v],n),\lambda^{u,v}(u,n)\}$$

In this paper, we compare the exact approach of Ten Eikelder et al. with the estimation approach of Balas and Vazacopoulos using our algorithm. The empirical testing shows that the estimation approach reduces the computational effort by 10 to 20 % in comparison with the exact approach, and the efficiency of the search increases as the instances become larger. Intuition suggests that exact approach might perform better than the estimation approach in the solution quality, but empirical results indicate that this intuition remains unconfirmed.

5.4 Tabu list and tabu status of move

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. The main purpose of using this attributive is to reduce the computational cost. A side effect of implementing the "a partial attribute" tabu list is that it may lead to giving a tabu status to unvisited solution or even an interesting solution. However, an aspiration criterion, which accepts the move provided that its makespan is lower than that of the current best solution found so far, is used by tabu search algorithm 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 tabu searches for the JSSP which only store the sequence of the operations exchanged in the tabu list, tabu search algorithm in this paper 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* (*u*,..,*w*,..,*v*) and their positions on the machine from *u* to *v* (p_u,..,p_w,..,p_v)) is not permitted for the duration that the move is recorded as "tabu".
The length of the tabu list determines the time limit of the moves remaining on the memory, which is discouraged at the current iterations. Therefore, the length of the tabu list plays an important role in the search process. Moreover, if the length of list is too short cycling cannot be avoided; conversely, a too long size creates too many restrictions and influences the intensification of the search. It has been observed that the average number of the visited solutions grows with the increase of the tabu list size. How to effciently set the length of tabu list in JSSP is still open problems, and setting the parameters often suffers from tedious trial and error. An empirical study suggests it may be possible to obtain superior results when the length of tabu list is allowed to vary dynamically during the course of the search. For example, Taillard (1994) suggests the length of the tabu list be randomly selected from a range between given minimal and maximal values and changed each time a number of iterations. Another example is the somewhat more sophisticated approach of Dell'Amico and Trubian (1993).

Therefore, 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 for the considered problem. 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 JSSP.

5.5 The recovery of the elite solutions based on the simulated annealing

In this paper, a powerful recency-based memory mechanism, which utilizes simulated annealing to find the elite solutions inside big valley, is built in the core tabu search and induces the search to pursue different trajectories. The recency-based memory mechanism is adjusted as follows. If the current solution satisfies:

$$f(s^*) < f(s^b) \text{ or } \exp(f(s^b) - f(s^*))/T > \text{random } [0, 1] \quad (2)$$

$f(s^*)$ 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 schedule and installed as the current solution. The TSSA algorithm then reinitiates the core tabu search 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 simulated annealing are stored in the elite solution stack L. The maximum number of the solution stack L is a fixed number and denoted as *Maxelite*. For the test instances, a *Maxelite* of 30 is found to offer an appropriate value. Temperature T of simulated annealing 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 solution stack being quickly exhausted, whereas if the temperature is too high, it can not guarantee that the elite solutions are effectively selected. Empirical testing shows that the suitable temperature T increases as the instances size becomes larger. For the general JSSP instances, T can be set to value 2-6 according to the instance size. It can be set to $T = bestMakespan/Temp$, 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 simulated annealing performs a "fine" search around 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, the TSSA algorithm converts the semi-active schedules popped from the elite solution stack into active schedules. This approach could better direct the search to explore new promising regions and hopefully increase the chances of finding the global optimum.

5.6 Move selection

The choice rule of the tabu search 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 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 TASA algorithm.

5.7 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 & Smutnicki (1996) and Jain et al. (2000). 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 search to escape from the cycling.

5.8 Termination criterion

The algorithm stops when it 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.

6. Computational results

The proposed algorithm above was implemented in VC++ language on a personal computer Pentium IV 3.0G. In order to evaluate and compare the performance of this algorithm, we tested it on the 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 $n×m$ = 6×6, 10×10, 20×5) due to Fisher and Thompson (1963), forty instances LA01-40 (size $n×m$ = 10×5, 15×5, 20×5, 10×10,15×10, 20×10, 30×10, 15×15) due to Lawrence (1984), five instances ABZ5-9 (size $n×m$ = 10×10, 20×15) due to Adams et al. (1988) and ten instances ORB01-10 (size $n×m$ = 10×10) due to Applegate & Cook (1991).
(b) Four instances denoted as YN1-4 (size $n×m$ = 20×20) due to Yamada & Nakano (1992) and twenty instances SWV01-20 (size $n×m$ = 20×10, 20×15, 50×10, 50×10) due to Storer et al. (1992).
(c) Eighty instances denoted as DMU01-DMU80 (size $n×m$= 20×15, 20×20, 30×15, 30×20, 40×15, 40×20, 50×15, 50×20) due to Demirkol et al(1998).
The FT, LA, ABZ, SWV and YN problems are available from the OR Library site http://www.ms.ic.ac.uk/job/pub/jobshop1.txt, while the DMU problems are available from http://gilbreth.ecn.purdue.edu/~uzsoy2/benchmark/problems.html. For these benchmark set, the best known upper bounds (UB_{best}) and the best known lower bounds (LB_{best}) are taken from Jain et al. (1999) and updated with the improved results from Nowicki & Smutnicki (2002).
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 $RE = 100×(UB_{solve} − LB_{best}) / LB_{best}$ for each instance. Due to the stochastic properties of this algorithm, it is not reasonable to compare the best makespan MRE (b-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.

In the remainder of this section, the first section gives the comparison of the neighborhood strategies and the move evaluation strategies respectively, in order to analyze the new neighborhood structure and the estimation strategy used in this paper. In the second section the proposed TSSA algorithm is performed on a large number of the benchmark instances to measure the performance of this algorithm.

6.1 Comparison of the neighborhood structures and move evaluation strategies

In order to compare the neighborhood structures and move evaluation strategies effectively, we set the length of elite solution stack L (*Maxelite*) to 0, and convert the TSSA algorithm to the traditional tabu search algorithm.

(1) Comparison of the neighborhood structures

Six neighborhood structures are tested and compared below. They are denoted by NS1 (van Laarhoven et al. 1992, N1), NS2 (Chambers & Barnes, 1994), NS3 (Nowicki & Smutnicki 1996, N5), NS4 (modify the Dell'Amico & Trubian, 1993, N4), NS5 (Balas & Vazacopoulos 1998, N6). Finally, NS6 denotes our new neighborhood structure. Meanwhile, NS1, NS2, NS3 and NS5 are well-known neighborhoods in literature, and these neighborhoods are all based on the concept of block except for N1. For further information, see Section3 and Blażewicz et al. (1996). NS4 slightly differs from N4 introduced by Dell'Amico and Trubian. NS4 neighborhood moves the operations in a block to the beginning or the end of this block according to the procedure of Dell'Amico & Trubian (1993), and the moves do not allow for an interchange on u and v when the critical path containing u and v contains neither $JS[u]$ nor $JP[v]$.

In order to evaluate these neighborhood structures exactly, we use tabu search algorithm as the platform, and the difference only exists among the neighborhood structures. The initial solution is generated by SPT priority dispatch rule, and the length of tabu list is set to 12 suggested by Geyik & Cedimoglu (2004) except for NS1 which applies the approach of Taillard. Each move in the neighborhood is evaluated exactly. The algorithm is terminated when the number of disimproving moves reaches to the value 3000. The benchmark set FT, LA, ABZ, SWV and YN containing 72 instances are tested. Moreover, several measures that gain some statistics relating to the comparison are presented. They are the mean makespan C_{max} (MC_{max}), the number of the solution found equal to the known best solution (NBE), the mean number of evaluated neighbors (MEN), the mean number of iterations (MNI) and the total CPU time performed in all instances (CPU-time).

	NS6	NS5	NS4	NS3	NS2	NS1
MC_{max}	1416	1420	1415	1462	1473	1600
MRE	3.31	3.51	3.45	5.52	5.84	12.6
NBE	32	33	32	26	24	20
MEN	106163	70216	127140	48973	48510	107649
MNI	5731	5468	5619	6690	6206	4149
CPU-time	309	205	462	121	134	307

Table 2. The comparison of the six neighborhood structures

Table 2 summarizes the computational results relating to the six neighborhoods. NS6 offers the minimum MRE value among the six neighborhood structures. The MRE provided by NS4 (better than the original N4) is close to that of NS6, but NS4 consumes too much computing times. NS5 is more effective than NS3, NS2 and NS1, and NS3 is better than NS2 and NS1. Overall, it can be seen that NS6 is an effective neighborhood structure for the JSSP.

(2) Comparison of the move evaluation strategies

The estimation approach proposed by Balas and Vazacopoulos and the exact evaluation approach suggested by Ten Eikelder et al. are compared using the tabu search algorithm as the test platform, see Table 3. The two test platforms are all similar except for the move evaluation methods. To make a full statistical comparison, all sets of benchmarks applied in this paper are selected and used to test. Table 3 presents the results of the estimation approach (Balas and Vazacopoulos) and the exact approach (Ten Eikelder et al.) when initiated from SPT priority rule.

	MC_{max}	MRE	NBE	MEN	MNI	CPU-time	Tabu list size
estimation approach	3340	4.42	40	215598	10445	367.2	12
exact approach	3339	4.44	41	234598	10785	2206.8	12

Table 3. Comparison of the estimation approach (Balas) with the exact approach (Ten Eikelder)

The empirical results in Table 3 indicate that the estimation approach of Balas and Vazacopoulos is about 5-6 times faster on average in evaluating moves in comparison to the exact evaluation of Ten Eikelder et al. In the experiments, the estimation approach performs better than the exact approach, by not only getting to the solutions faster, but also having no effect on the solution quality. For example, these two evaluation strategies provide the approximately similar value of MRE. However, it can be seen from Table 3 that the estimation approach need the relatively "small" MNI (the mean number of iterations) to achieve the similar value of MRE. A plausible explanation for this is that applying the estimation strategy leads to the algorithm to intensify search in the visited region and mitigates the drawback of the estimation approach. Therefore, the estimation approach proposed by Balas and Vazacopoulos is implemented to perform these computations of each move in the following section.

6.2 TSSA algorithm for the JSSP

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 Taboo Search of Nowicki & Smutnicki (1996), BV stands for the Guided local Search with Shifting Bottleneck of Balas & Vazacopoulos (1998). Meanwhile, SB-RGLS5 stands for the solution of SB-RGLS5 procedure of Balas & Vazacopoulos (1998) and BV-best stands for the best solution obtained by Balas & Vazacopoulos. TSSB stands for a tabu search method guided by shifting bottleneck of Pezzella & Merelli (2000). Among these papers listed above, the algorithms TSAB, BV and TSSB provide the detailed makespan and running time of each instance.

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. Nevertheless, 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.

TSSA algorithm was initiated from the active solution randomly generated. Parameters of the length of tabu list, *ImproveIter* (limit on umimproved iterations), *T* (Tempeature) and *TotIter* (the total number of the iterations) were chosen experimentally in order to ensure a compromise between efficiency and effectiveness. The choice of the length of tabu list can be seen in the Section 5.4. Parameter *ImproveIter* was bounded by the given minimal and maximal values between 2500-5000 depending on the number of jobs (n) and the number of machines (m). 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 5.5. Meanwhile, if the number of operations is less than or equal to 400, *Temp* = 300. Otherwise, *Temp* = 300+50×n/m. The parameter *TotIter* has an influence on the running time and 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 solution quality, our preliminary experiments show that *TotIter* can be set to $n\times100000/2$ for general hard instances, but for the particularly hard instances it increases correspondingly for the sake of finding a better solution.

During various tests (with tuning parameters) and standard tests, TSSA algorithm found some new upper bounds. The best upper bounds may be found after the running of many times, but standard tests only ran ten times to get the average makespan and running time of each instance.

(1) 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 4 shows the comparison of the performance of TSSA algorithm with those of TSAB, BV-best and TSSB. In this table, it 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 4 (similar to Table 5-8 below). It can be seen that TSSA algorithm performs very quickly and acquires the results in only half a 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.

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 133MHz.

Table 4. Comparison with the other three algorithms for LA and ABZ instances

Problem	Size	UB(LB)	TSSA			TSAB	BV		TSSB
			Best	M_{av}	$T_{av}(s)$		SB-RGLS5	BV-best	
FT10	10×10	930	930*	930	3.8	930	930	930	930
LA19	10×10	842	842*	842	0.5	842	842	842	842
LA21	15×10	1046	1046*	1046	15.2	1047	1046	1046	1046
LA24	15×10	935	935*	936.2	19.8	939	935	935	938
LA25	20×10	977	977*	977.1	13.8	977	977	977	979
LA27	20×10	1235	1235*	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*	1268	9.9	1268	1268	1268	1268
LA37	15×15	1397	1397*	1402.5	42.1	1407	1397	1397	1411
LA38	15×15	1196	1196*	1199.6	47.8	1196	1196	1196	1201
LA39	15×15	1233	1233*	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**[d]	684.8	90.2	695	679	679	693
MRE			0.43	0.67	–	1.04	0.61	0.53	1.09

* The best solutions found by our algorithm are equal to the best known lower bounds

[d] The best makespans our algorithm found are better than the best previously known values

Table 5. Results for the fifteen tough instances

To make a more detailed performance comparison of the TSSA algorithm with the other algorithms, we select the fifteen most difficult instances among FT, LA and ABZ benchmarks. The majority of the 15 instances have been viewed as computational challenges, and even the optimal solutions of the ABZ8 and ABZ9 instances remain unknown until now. Table 5 shows the makespan performance statistics of each algorithm for the fifteen difficult problems, and Fig. 9 illustrates the gantt chart of the optimum solution for LA38 (15×15).

In this table, the column named UB (LB) lists the best known upper bounds(lower bounds) indicates in Jain et al. (1999), 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 mean relative error (MRE) in order to analyze the effectiveness of these algorithms.

Table 5 shows that the av-MRE provided by TSSA is lower than the MREs 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 finds 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 of finding 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 solution. 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—678.

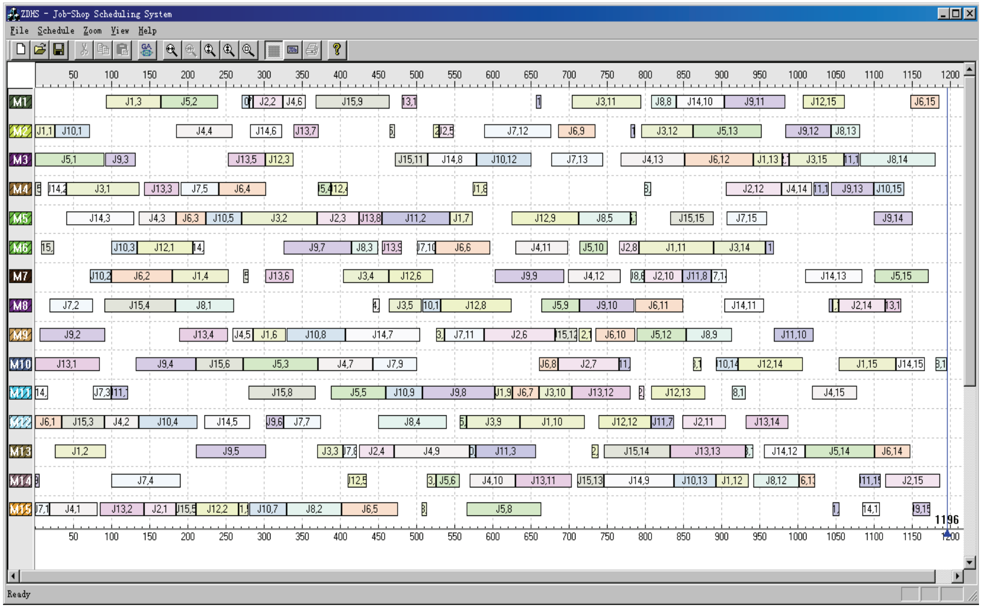
ZDHS - Job-Shop Scheduling System
File Schedule Zoom View Help
M1
M2
M3
M4
M5
M6
M7
M8
M9
M10
M11
M12
M13
M14
M15
1196
Ready

Fig. 9. Gantt chart showing the optimum solution for LA38 (15×15)

Finally, the ORB class which contains 10 instances is analyzed. Table 6 lists the detailed results of comparison. In this table, TSSA is clearly better than TSAB, BV-best and TSSB in terms of 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.

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

Table 6. Results for ORB01-10 instances

(2) Results for instances (b)

YN class contains 4 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. Benchmarks YN and SWV have not been tested by algorithm TSSB. Therefore we primarily compare TSSA with SB-RGLS5 and BV-best.

Table 7 shows the detailed results of comparison for YN instances. The solution quality TSSA algorithm provides is clearly better than that of BV in a short time. For example, TSSA algorithm achieves av-MRE = 6.99% within two minutes on our personal PC, whereas BV-best needs approximately 150 minutes on the SUN SParc-330 to achieve the similar aim. Moreover, TSSA algorithm found two new upper bounds, namely: YN1—884 and YN2—907. Fig. 10 illustrates the gantt chart of the best solution whose makespan is equal to 884 for YN1.

Similarly as for YN instances, the detailed results for SWV instances are shown in Table 8. In order to find a better solution, *TotIter* is set to n×100000 for SWV instances. The solution quality TSSA algorithm provides outperforms that of BV in a short time. For example, for SWV06-10 instances, TSSA algorithm offers b-MRE = 6.91% in about three minutes on the personal PC, whereas BV-best provides b-MRE = 8.11% in approximately 180 minutes 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.

			TSSA			BV[b]			
Problem	Size	UB(LB)	Best	M_{av}	$T_{av}(s)$	SB-RGLS5	CPU(s)	BV-best	CPU(s)
YN1	20×20	885(826)	**884**[d]	891.3	106.3	893	3959.2	891	9382.4
YN2	20×20	909(861)	**907**[d]	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

Table 7. Results for YN1-4 instances

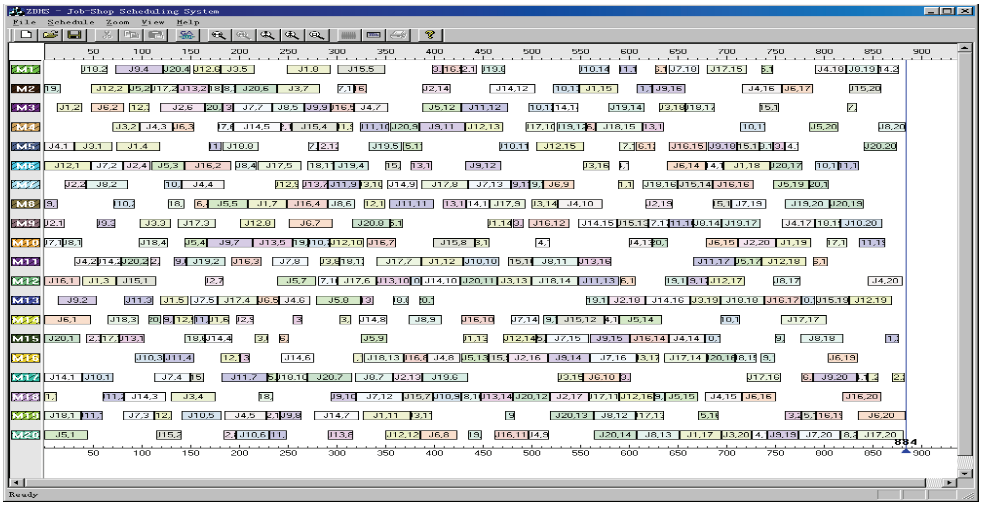

Fig. 10. Gantt chart showing the best solution (makespan is equal to 884) for YN1

			TSSA			BV[b]			
Problem	Size	UB(LB)	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*	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**[d]	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**[d]	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**[d]	1783.7	184.6	1794	3005.2	1773	10332
SWV06-10			6.91	8.66	190.2	9.27	4294.7	8.11	11084

Table 8. Results for SWV01-10 instances

7. Conclusion

The efficiency of the tabu search for the JSSP depends on the neighborhood structures and initial solution. In this paper, firstly, a new neighborhood structure is constructed, which could investigate much larger solution space. We compare the new neighborhood structure with the other five neighborhood strategies, and confirm that it is an effective neighborhood structure for the JSSP. Furthermore, the effects of neighborhood evaluation strategies are investigated. Empirical testing discloses that the estimation approach introduced by Balas and Vazacopoulos not only significantly improves the efficiency of the search, but also has no material effect on the solution quality.

Secondly, through the proper use of the structure of the solution space (especially the big valley), we developed the novel hybrid TSSA algorithm which combines the advantage properties of simulated annealing with tabu search strategy. This algorithm mitigates the drawback of tabu search and could reduce the influence of the initial solution and obtain the high-quality solutions in a short running time on a modern PC, which have been confirmed by tests on a large number of benchmark problems. Moreover, it improves a lot of the current best solutions with 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 tabu search and simulated annealing could also be applied to solving the other difficult combinatorial optimization problems.

In addition, we observed that the TSSA algorithm has better efficiency for the JSSP than the traditional tabu search when $n \leq 2m$; nevertheless, for some rectangle problems ($n >> m$) tabu search with the powerful neighborhood structures and dynamic tabu list could be more effective than the hybrid tabu search approach. Therefore, a subject of future work would exploit the more effective diversification strategy of hybrid tabu search. In addition, how to efficiently set the tabu list in JSSP is still an open problem. The growing researches suggest that the length of short term memory varies dynamically in response to the changing conditions of the search, but there is still a broad research for better implementations.

8. Reference

Adams, J.; Balas, E. & Zawack, D. (1988) The shifting bottleneck procedure for job shop scheduling. Management Science, Vol. 34, pp. 391–401.

Balas E. (1969) Machine sequencing via disjunctive graph: an implicit enumeration algorithm. Operations Research, Vol.17, pp. 941-57.

Balas, E. & Vazacopoulos, A. (1998) Guided local search with shifting bottleneck for job shop scheduling. Management Science, Vol. 44, No. 2, pp. 262–75.

Barnes, J.W.; & Chambers, J.B. (1995) Solving the job shop scheduling problem using tabu search. IIE Transactions, Vol. 27, pp. 257–63.

Blażewicz, J.; Domschke, W. & Pesch, E. (1996) The job shop scheduling problem: Conventional and new solution techniques. European Journal of Operational Research, Vol. 93, pp. 1–33.

Brucker, P. Scheduing algorithm. Berlin: Springer-Verlag, 1995.

Brucker, P.; Jurisch, B. & Sievers, B. (1994) A branch and bound algorithm for job-shop scheduling problem. Discrete Applied Mathematics, Vol. 49, pp. 105–27.

Carlier, J. & Pinson, E. (1989) An algorithm for solving the job shop problem. Management Science, Vol. 35, No.29, pp. 164–76.

Chambers, J.B. & Barnes, J.W. (1996) 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,.

Croce, F.D.; Tadei, R. & Volta, G. (1995) A genetic algorithm for the job shop problem. Computers & Operations Research, Vol. 22, pp. 15–24.

Dell'Amico, M. & Trubian, M. (1993) Applying tabu-search to job-shop scheduling problem. Annals of Operations Research, Vol. 41, No. 1–4, pp.231–52.

Demirkol, E. Mehta, S. & Uzsoy, R. (1998) Benchmarks for shop scheduling problems. European Journal of Operational Research, Vol. 109, pp. 137-41.

Garey, E.L., Johnson, D.S. & Sethi. R. (1976) The complexity of flowshop and job-shop scheduling. Mathematics of Operations Research, Vol. 1, pp. 117–29.

Geyik, F. & Cedimoglu, I.H. (2004) The strategies and parameters of tabu search for job-shop scheduling.Journal of Intelligent Manufacturing, Vol. 15, pp. 439–48.

Giffler, B. & Thompson, G.L. (1960) Algorithms for solving production scheduling problem. Operations Research, Vol. 8, No. 4, pp. 487-503

Glover, F. & Hanafi, S. (2002) Tabu search and finite convergence. Discrete Applied Mathematics, Vol. 119, No. 1-2, pp. 3–36.

Glover, F. & Laguna, M. (1997) Tabu search. Dordrecht: Kluwer Academic Publishers.

Glover, F. (1989) Tabu search—Part I. ORSA Journal on Computing, Vol. 1, No. 3, pp. 190-206.

Glover, F. (1990) Tabu search—Part II. ORSA Journal on Computing, Vol. 2, No. 1, pp. 4-32.

Hanafi, S. (2000) On the convergence of tabu search. Journal of Heuristics, Vol. 7, pp. 47-58.

Hansen P. (1986) The steepest ascent mildest descent heuristic for combinatorial programming. in: Congress on Numerical Methods in Combinatorial Optimization, Capri, Italy.

Jain, A.S. & Meeran, S. (1999) Deterministic job shop scheduling: past, present and future. European Journal of Operational Research, Vol. 113, pp. 390–434.

Jain, A.S.; Rangaswamy, B. & Meeran, S. (2000) New and "stronger" job-shop neighbourhoods: A focus on the method of Nowicki and Smutnicki (1996). Journal of Heuristics, Vol. 6, No.4, pp. 457–80.

Lawrence, S. (1984) Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques (Supplement). Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania,

Matsuo, H.C.; Suh, C.J. & Sullivan, R.S. (1988) A controlledsearch simulatedannealing methodfor general job shop scheduling problem. Working Paper, 03-04-88, Department of Management, The University of Texas at Austin, Austin, TX.

Muth, J.F. & Thompson, G.L. (1963) Industrial scheduling. Englewood Cliffs, NJ: Prentice Hall.

Nowicki E, Smutnicki C. (2002) Some new tools to solve the job shop problem. Technical Report 60/2002, Institute of Engineering Cybernetics, Technical University of Wroclaw, Wroclaw, Poland.

Nowicki, E. & Smutnicki, C. (1996) A fast taboo search algorithm for the job shop scheduling problem. Management Science, Vol. 42, No. 6, pp. 797–813.

Nowicki, E. & Smutnicki, C. (2001) Some new ideas in TS for job-shop scheduling. Technical Report 50/2001, Institute of Engineering Cybernetics, Wroclaw University of Technology, Wroclaw, Poland.

Nowicki, E. & Smutnicki, C. (2002) Some new tools to solve the job shop problem. Technical Report 60/2002, Institute of Engineering Cybernetics, Technical University of Wroclaw, Wroclaw, Poland.

Pezzella, F. & Merelli, E. (2000) A tabu search method guided by shifting bottleneck for job shop scheduling problem. European Journal of Operational Research, Vol. 120, pp. 297–310.

Storer, R.H.; Wu, S.D. & Vaccari, R. (1992) New search spaces for sequencing problems with applications to job-shop scheduling. Management Science, Vol. 38, No. 10, pp. 1495-1509.

Taillard, É.D. (1989) Parallel taboo search technique for the job shop scheduling problem. Technical Report ORWP 89/11, DMA, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland.

Taillard, É.D. (1994) Parallel taboo search techniques for the job-shop scheduling problem. ORSA Journal on Computing, Vol. 6, pp. 108–17.

Ten Eikelder, H.M.M.; Aarts, B.J.M; Verhoeven, M.G.A. & Aarts, E.H.L. (1999) Sequential and paralled local search algorithms for job shop scheduling. In: Voss S, Martello S, Osman IH, Roucairol C, editors. Meta-Heuristic:advances and Trends in Local Search Paradigms for Optimization. Massachusetts: Kluwer Academic Publishers, pp. 359–71.

Vaessens, R.J.M.; Aarts, E.H.L. & Lenstra, J.K. (1996) Job shop scheduling by local search. INFORMS Journal on Computing, Vol. 8, pp. 302–17.

Van Laarhoven, P.J.M.; Aarts, E.H.L. & Lenstra, J.K. (1992) Job shop scheduling by simulated annealing. Operations Research, Vol. 40 No. 1, pp. 113–25.

Yamada, T. & Nakano, R. (1992) A genetic algorithm applicable to large-scale job-shop problems. in: Manner R, Manderick B (Eds.). Proceedings of the Second International Workshop on Parallel Problem Solving from Nature (PPSN'2), Brussels, Belgium, pp.281-290.