JOURNAL OF INDUSTRIAL AND
MANAGEMENT OPTIMIZATION
Volume **12**, Number **2**, April **2016**

doi:10.3934/jimo.2016.12.703

pp. **703–717**

A TABU SEARCH ALGORITHM TO MINIMIZE TOTAL WEIGHTED TARDINESS FOR THE JOB SHOP SCHEDULING PROBLEM

Y. K. Lin

Department of Industrial Engineering and Systems Management,
Feng Chia University, P.O. Box 25-097,Taichung, 40724, Taiwan, ROC

C. S. Chong

Planning and Operations Management Group
Singapore Institute of Manufacturing Technology
A-star, 7 Nanyang Avenue, 638075, Singapore

(Communicated by Joseph Leung)

Abstract. This research presents a tabu search algorithm with a restart (TSA-R) approach to minimize total weighted tardiness (TWT) for the job shop scheduling problem. Jobs have non-identical due dates. The problem belongs to the class of NP-hard problems. The TSA-R approach uses dispatching rules to obtain an initial solution and searches for new solutions in a neighborhood based on the critical paths of jobs and blocks of operations. The TSA-R applies a new diversification scheme to exploit the initial solutions and its neighborhood structures so as to overcome entrapment issues and to enhance solutions. A computational result based on standard benchmark instances from the literature is presented to show the effectiveness of the proposed tabu search algorithm.

1. **Introduction.** Meeting due dates is a major issue in most manufacturing systems, and one effective measure for due dates is total weighted tardiness. Tardiness is crucial in manufacturing systems, since whenever a job is not completed by its due date, certain direct or indirect costs are incurred. These costs include penalty clauses in contracts, loss of goodwill and customers, and a damaged reputation.

A job shop is a generalized production system with distinct machines in the shop. In a job shop environment, each job requires some of these machines in some specific sequence. Some industry problems that are classified under the structure of the general job shop include the processing of different batches of crude oil at a refinery, the repair of cars in a vehicle workshop, or the manufacturing of different paint colors. A special case of job shop environment can be described by a set of n jobs to be processed through a set of m machines, with each job consisting of an ordered sequence of m operations. The processing of a job on a machine is called an operation; each operation is processed for a duration called the processing time. Each machine can process only one operation at a time, and preemption is not permitted. The job shop problem is to find a schedule to minimize one or more criteria such as makespan, total weighted tardiness, and maximum tardiness. The

2010 *Mathematics Subject Classification.* Primary: 90B35; Secondary: 68M20.
Key words and phrases. Scheduling, tabu search, job shop, total weighted tardiness.

problem of minimizing tardiness in job shops is strongly NP-hard (Lawler *et al.*, [16]). The objective of this research is to determine a schedule that minimizes total weighted tardiness pertaining to job due dates for the special case of job shops. Total weighted tardiness is defined as $\sum w_j T_j$, where $T_j = max(C_j - d_j, 0)$, C_j is the completion time of job j, w_j and d_j are the weight and due date of job j , respectively. Total weighted tardiness is a measure of customer satisfaction. Minimizing it represents satisfying the general requirement of on-time-delivery. Following the three-field notation of Graham *et al.* [11], we refer to this problem as $J_m||\sum w_j T_j$.

The rest of the article is organized as follows. In the next section, we present a review of the literature related to this article. Section 3 gives the representation of the job shop scheduling problem. In section 4, the proposed tabu search algorithm is presented. A description of intensification and diversification with elite solutions follows in section 5. In section 6, the computational results are reported. Section 7 presents our conclusions and some suggestions for future research.

2. **Literature review.** For job-shop-related tardiness problems $(J_m||\sum T_j)$, dispatching rules are commonly used because they are simple to implement and quick to execute (Kanet and Hayya, [14]; Baker, [4]). However, their performance is generally unpredictable. Some experimental studies (Vepsalainen and Morton, [27]) on dispatching rules have shown that the Apparent Tardiness Cost (ATC) rule achieves the best results. Baker and Kanet [5] used a modified operation due date (MOD) rule for mean tardiness problems in job shops. Anderson and Nyirendra [1] also proposed two new dispatching rules closely related to the MOD rule for minimizing tardiness in a job shop. Armentano and Scrich [2] proposed several heuristics and found that the MDD (modified due date) rule provided the best solutions. Recently, elaborate heuristic methods have been proposed for $J_m||\sum T_j$ problems. Raman and Talbot [24] developed a specific heuristic approach to construct a schedule by focusing on bottleneck machines. He *et al.* [12] presented a multi-pass heuristic algorithm. Yang *et al.* [28] developed a revised exchange heuristic algorithm (REHA). The algorithm was shown to minimize total tardiness notably for problems of practical size. Singer and Pinedo [25] used branch and bound techniques for the total weighted tardiness problem in job shops subject to release dates $(J_m|r_j|\sum w_j T_j)$. Pinedo and Singer [23] applied a shifting bottleneck heuristic (SBH) for the $J_m|r_j|\sum w_j T_j$ problem. Kreipl [15] proposed a large step random walk for minimizing total weighted tardiness in a job shop. Asano and Ohta [3] proposed a heuristic based on a tree search for the $J_m||\sum w_j T_j$ problem. Mati *et al.* [18] proposed a general approach for optimizing regular criteria in the job shop problem ($J_m|r_j|C_{\max}$,$J_m|r_j|\sum w_j C_j$,$J_m|r_j|\sum w_j T_j$,$J_m|r_j|\sum w_j U_j$). Lu *et al.* [17] have experimented on the application of order review/release mechanisms combined with dispatching rules in assembly job shop scheduling with respect to mean absolute deviation and mean shop floor through put time measures. Nguyen *et al.* [22] investigated the use of genetic programming as a hyper-heuristic method for automatically discovering new dispatching rules for $J_m||C_{\max}$ and $J_m||\sum w_j T_j$ problems. Calleja and Pastor [9] presented a dispatching rule to solve a real-world case of flexible job shop scheduling problem with transfer batches to minimizing the average tardiness of production orders.

Recently, general-purpose metaheuristics such as tabu search algorithms, genetic algorithms, simulated annealing, and bee colony algorithms have been applied to job shop scheduling with tardiness objectives. Armentano and Scrich [2] applied a tabu search approach for the $J_m||\sum T_j$ problem. Mattfeld and Bierwirth [19] considered a genetic algorithm for job shop scheduling problems with release dates, due dates, and various tardiness objectives. Bontridder [7] considered a tabu search algorithm for the $J_m|r_j|\sum w_jT_j$. Zhang and Wu [33] applied a simulated annealing algorithm which utilized a block-based neighborhood structure to solve the $J_m||\sum w_jT_j$ problem. Different approaches can also be combined to improve job shop heuristics. For example, for the $J_m|r_j|\sum w_jT_j$ problem, Essafi *et al.* [10] proposed a genetic algorithm with an iterated local search that used a longest path approach on a disjunctive graph model. Zhou *et al.* [35] studied the $J_m||\sum w_jT_j$ problem. They used a genetic algorithm to determine the first operation of each machine, and a heuristic to determine the assignment of the remaining operations. Zhang and Wu [31] used a divide-and-conquer strategy with particle swarm optimization for the $J_m||\sum w_jT_j$ problem. Zhang and Wu [32] presented a hybrid immune simulated annealing algorithm for the $J_m||\sum w_jT_j$ problem. Zhang [30] used a genetic local search algorithm based on insertion neighborhood for the $J_m||\sum w_jT_j$ problem. Bulbul [8] proposed a hybrid shifting bottleneck-tabu search algorithm for the $J_m||\sum w_jT_j$ problem, in which the shifting bottleneck algorithm's re-optimization step was replaced by a tabu search. Zhang *et al.* [34] applied a hybrid artificial bee colony algorithm for the $J_m||\sum w_jT_j$ problem.

The objective of our research is to determine a schedule that minimizes total weighted tardiness pertaining to job due dates in classical job shops ($J_m||\sum w_jT_j$). We approach this problem by means of tabu search metaheuristics, which are known to perform well for minimizing makespan in job shops. Our methods start from initial solutions obtained through dispatching rules and then use tabu search to explore new and better neighborhood solutions.

3. **Representation of the job shop problem.** The job shop scheduling problem can be represented with a type of disjunctive graph introduced by Balas [6]. A similar graph was implemented by Pinedo and Singer [23] for the $J_m||\sum w_jT_j$ problem. A disjunctive graph $G = (N, A, E)$ is defined as follows: N is $\{S, 1,2,\ldots, D_j\}$ the set of nodes representing all operations where S and D_j represent one source node and n sink nodes, respectively. A is the set of conjunctive (directed) arcs that connect the nodes; each arc in A represents a pair of consecutive operations of the same job. E is the set of disjunctive arcs that connect operations to be processed by the same machine. $E = \bigcup_{k=1}^{m} E_k$, where E_k is the subset of disjunctive pair-arcs corresponding to machine k. Let π denote a selection of disjunctive arcs from E. π contains exactly one directed arc between each pair of oppositely directed arcs in E such that the resulting graph $G = (\pi)$ is acyclic. The processing order π is feasible only if graph $G = (\pi)$ does not contain a cycle. Here, we use an example of three jobs and three machines given in Table 1 to illustrate our tabu search method. This problem can be represented by a disjunctive graph shown in Fig.1, where $E_1 = \{2,4,7\}$, $E_2 = \{3,5,9\}$, and $E_3 = \{1,6,8\}$. A feasible solution for the disjunctive graph in Fig.1 is shown in Fig.2, where a selection of disjunctive arcs from E (π) is $4\to7\to2$ for E_1; $5\to3\to9$ for E_2; $1\to8\to6$ for E_3. A Gantt chart corresponding to Fig.2 is shown in Fig.3.

Example 1: This problem has 3 jobs, 3 machines and 9 operations. The routes of the jobs, and the processing times are given in the following table.

TABLE 1. Data for Example 1

Job	w_j	d_j	Machine sequence	Operations	Processing time
1	2	9	3,1,2	1,2,3	$p_{31} = 2, p_{11} = 4, p_{21} = 1$
2	1	8	1,2,3	4,5,6	$p_{12} = 3, p_{22} = 5, p_{32} = 2$
3	3	10	1,3,2	7,8,9	$p_{13} = 1, p_{33} = 2, p_{23} = 3$

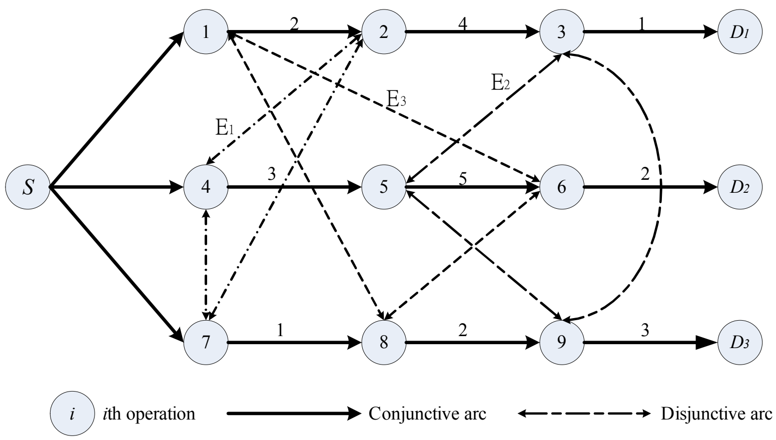

FIGURE 1. Disjunctive graph $G = (N, A, E)$ for Example 1

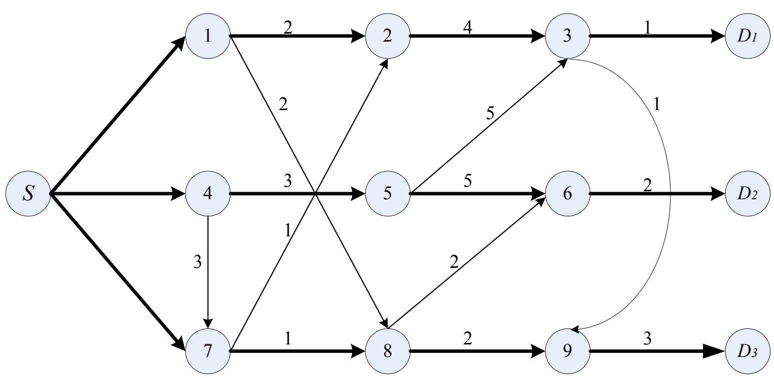

FIGURE 2. A feasible solution $G = (\pi)$ for the disjunctive graph in Fig.1

4. **Tabu search algorithm for the job shop problem.**

4.1. **The neighborhood structures.** Critical paths play an important part in a feasible solution. Critical paths are the longest routes from start node S to destination nodes D_j in a directed graph $G = (\pi)$. The completion time C_j of job j is equal to the longest route from S to D_j. A block is a maximal subsequence of operations in a critical path which contains operations processed on the same machine. For example, in Fig.3, jobs 1 and 3 have two critical paths and job 2 has only one critical path. Job 1: (4,7,2,3) and (4,5,3); job 2: (4,5,6); job 3: (4,7,2,3,9) and (4,5,3,9). The completion times for the jobs are: $C_1 = 9$, $C_2 = 10$, and $C_3 =$

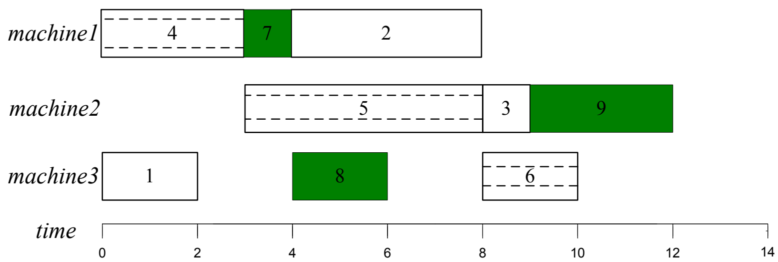

FIGURE 3. Gantt chart for the directed graph in Fig.2

12. Critical path (4,7,2,3) comprises two blocks B_1= (4,7,2), B_2= (3) and critical path (4,7,2,3,9) also comprises 2 blocks B_1= (4,7,2), B_2= (3,9).

A move is defined as a function which transforms a solution into another solution. All solutions obtained by applying moves on each critical path of a given job generate a neighborhood. There are various neighborhood structures defined in the literature. Recently, several well-known neighborhood approaches have used the concept of a block, in which a move is defined by changing the positions of operations in a block on a critical path. In this research, we adopted neighborhood structures proposed by Van Laarhoven *et al.* [26], denoted by N_1. N_1 is generated by swapping any adjacent pair of critical operations on the same block, as shown in Fig.4. The size of N_1 is quite large and the approach is time consuming. However, our preliminary result based on design of experiment shows that it tends to get better TWT than the other approach since it considers more moves.

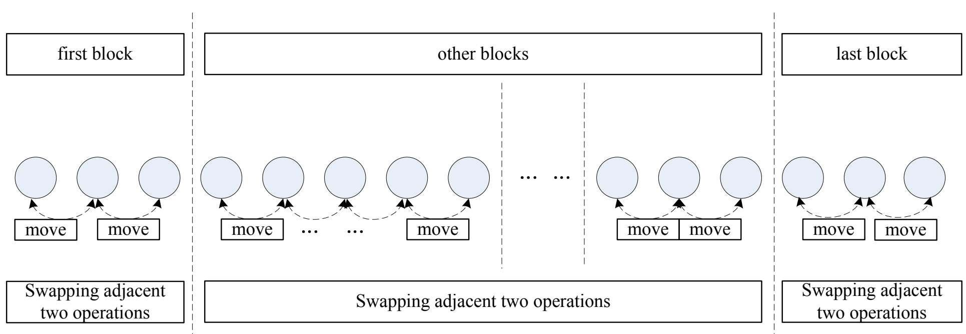

FIGURE 4. Van Laarhoven *et al.* [26] neighborhood of moves (N_1)

4.2. **Neighborhood searching procedure.** Let $L_{TB} = (L_1, \ldots, L_{maxt})$ be the tabu list of a fixed length *maxt* that saves forbidden moves, and let TWT* be the best known total weighted tardiness. Once a move is performed on the processing order, it will be added into L_{TB}. Let *Iter* represents the number of iterations to run and *NonImprovingIter* represents the number of non-improving iterations to run. *CPType* denotes the percentage of the total number of jobs, ranked in non-increasing order of TWT to be considered in a tabu search. It will scale with the size of job shop problem. *CPRandomType* defines the percentage number of jobs. It will scale with the size of the job shop problem and introduce a stochastic element into the tabu procedure. Our neighborhood search procedure can be described as follows.

Step 1. Generate several initial solutions by dispatching rules. Select the best one as the initial solution.
Step 2. Initially, set *IterCnt*=0 and *NonImpIterCnt*=0.
Step 3. Set *IterCnt* = *IterCnt* +1.
Step 4. If *IterCnt*> *Iter*, then terminate the neighborhood search procedure; otherwise, let WT be the ordered set of all jobs, ordered by non-increasing weighted tardiness ($w_j T_j$) from a given feasible solution.
Step 5. Choose the first *CPType* jobs from WT, and remove those jobs from WT. Choose another *CPRandomType* job randomly from the remaining jobs in WT. Find all their corresponding critical paths.
Step 6. Construct a neighborhood list of moves from the selected critical paths in Step 5, and store the distinct moves in list L_m.
Step 7. Sort L_m moves in non-increasing order of the largest critical jobs ($w_1 T_1 + w_2 T_2$).
Step 8. If all the moves in L_m are tabu moves, and performing those moves cannot improve the current TWT*, we use the same method, as proposed by Nowicki and Smutnicki [20] to modify the tabu list.
Step 9. Select and remove the first move M_i from $L_{m,}$. Construct a new schedule based on the move M_i. If the new constructed schedule TWT_i<TWT*, go to Step 10; otherwise, go to Step 11.
Step 10. Update TWT*= TWT_i, set *NonImpIterCnt* = 0. If M_i is not in L_{TB}, then add M_i into L_{TB}. Go back to Step 3 with the newly update schedule(solution).
Step 11. If M_i is a tabu move, then do not perform the move; otherwise perform the move and add M_i into L_{TB} and set *NonImpIterCnt*= *NonImpIterCnt*+1.
Step 12. If *NonImpIterCnt*>*NonImprovingIter*, then terminate neighborhood search procedure; otherwise, go back to Step 8.

In Step 1. we use WEDD (weighted earliest due date), MDD, ATC, and SPT (shortest processing time) rules to generate initial solutions. The initial solution with the lowest value of TWT will be used for subsequent procedure. These rules are derived from existing papers. In Step 2, we initialize iteration count *IterCnt*=0 and non-improving iteration count *NonImpIterCnt*=0. In Step 4, if *IterCnt*>*Iter*, then terminate the neighborhood search procedure; otherwise, we sort all jobs in non-increasing weighted tardiness ($w_j T_j$) from a given feasible solution. In Step 5, we choose the first *CPType* jobs from WT, and remove those jobs from WT. Next, we choose another *CPRandomType* job randomly from the remaining jobs in WT. This part is a refinement from authors. For each of these chosen jobs, we find all their critical paths.

In Step 6. we construct a neighborhood by using neighborhood structures N_1 for all critical paths identified in Step 5, and store the distinct moves in list L_m. In Step 7, since each neighbourhood move consists of two operations, each operation of the move belongs to a different job, and therefore we sort the moves in a non-increasing order of the largest critical jobs ($w_1 T_1 + w_2 T_2$). In Step 8, if all the moves in L_m are tabu moves and performing those moves cannot improve the current TWT*, we would modify tabu list. Specifically, we modify the tabu list by removing the oldest tabu move from the $L_{TB,}$ and replicate the youngest move in L_{TB} until *maxt* forbidden moves are saved in L_{TB}. We use the same example as in Nowicki and Smutnicki [20] to illustrate the modification. Suppos L_m ={(4,5),(1,2)} contains only tabu moves that cannot improve the current TWT*, and there is a tabu list $L_{TB} = \{(2,3),(1,2),(6,5),(4,5),(2,7)\}$ of length *maxt* = 5. The rule in Nowicki and Smutnicki [20] will select the move (1,2), and the modified tabu list will take the form of $L_{TB}= \{(6,5),(4,5),(2,7),(2,7),(2,7)\}$. In Steps 9-10, we select the first move M_i from L_m and remove M_i from L_m. If performing the move M_i can lead to a better TWT than the current TWT*, we would perform the move, even if it is a tabu move. After the move, we update the current TWT*= TWT$_i$, reset *NonImpIterCnt* =0, and go to next iteration. In Step 11, if M_i is a tabu move and performing the move M_i cannot lead to a better TWT than the current TWT*, we will not perform the move. However, if M_i is not a tabu move, we will still perform the move, even through it leads to a worse TWT than the current TWT*. Then, we update the tabu list and increment *NonImpIterCnt* by 1. In Step 12, if *NonImpIterCnt*>*NonImprovingIter*, we would terminate the neighborhood search procedure; otherwise, go back to Step 8.

5. Intensification and diversification with elite solutions.

5.1. Standard procedure.

The basic tabu search algorithm (TSA) described in section 4 is very computationally efficient. However, the best solutions found are in no way comparable to those of other local search methods. This performance issue can be addressed by introducing the concept of elite solutions. The TSA will then become a part of a more complex elite solution management system that can potentially lead to good regions of the solution set. Whenever the current solution is better than the best solution obtained up to that point, the current solution is marked as an elite solution, and its attributes are saved for later recovery. The elite solutions can be considered as a set of local optima from which a later recovery operation can jump start to find better minima. This solution recovery mechanism is termed Taboo Search Algorithm with Back Jump Tracking (TSAB) by Nowicki and Smutnicki [20]. This approach exploits the neighborhood by means of a tabu search intensification strategy that couples recency-based memory with the recovery of elite solutions. This search strategy intensifies the search in the vicinity of good local optima, and causes the search to pursue different trajectories. This solution recovery mechanism can reduce deviation from optimality.

The main idea of TSAB is related to a strategy that resumes the search from unvisited neighbors of the best previously-generated solutions. Every time a new best solution is found, that solution, together with its attributes, is stored as an elite solution. The maximum number of elite solutions that can be stored is *MaxElite*. A new elite solution will simply overwrite the poorest elite solution if *MaxElite* solutions have already been stored. After each discovery of a new best solution, the TSA will be run for a further *MaxIters* iterations (or *MaxNonImprovingIters*). *MaxIters* is a parameter that defines the number of iterations the algorithm will run after an elite solution has been found; *MaxNonImprovingIters* defines the maximum non-improving iterations that the algorithm will run after an elite solution has been found. The attributes of the solution are a set of next moves and the tabu list at the stage when the solution was saved. More specifically, the elite solutions procedure (TSAB) is as follows:

Step 1. Generate an initial solution, marking it as the current best and save it as an elite solution for later recovery.

Step 2. Perform neighborhood searching procedure (NSP) as defined in section 4.2.

Step 3. Check if the best solution has been improved: if yes, go to Step 4, otherwise go to Step 5.

Step 4. Save this solution and its attributes as an elite solution for later recovery.

Step 5. Go to Step 2 if any of the terminating conditions has not been met, otherwise go to Step 6.

Step 6. If there still exists any elite solution, recover the next elite solution and go to Step 2, otherwise STOP.

In Step 4, all the possible moves from the solution are stored except for the move from which the search continued at the previous point. These stored moves are ranked in order by non-decreasing total weighted tardiness, with tabu moves ranked after all the non-tabu moves. The search is always reinitiated from the first ranked move, which is then removed from the list so that the search will not be reinitiated from it again in a later recovery. This removal prevents cycling and encourages a different search trajectory.

Once a series of recovery operations has eliminated all moves for the elite solution, this solution is dropped from the elite solution list l, so that the next recovery starts from the next elite solution. Since l can also be changed whenever a new best solution is added to the list, each solution is recovered for a number of iterations determined by its quality with respect to the other stored elite solutions. If the current best solution is improved during a recovery operation, at iteration t, then the method is run until the iteration count equals *MaxIters* + t or *MaxNonImprovingIters*, before considering a recovery of the improved solution. The algorithm stops when all elite solutions on the list l have been reinitiated from all possible moves. For a more precise description of TSAB, refer to section 3.4 of the paper by Nowicki and Smutnicki [20].

5.2. **Enhanced recovery procedure with elite solutions.** The inherent weakness of many local search procedures is that they often get trapped in a region around some local optima. Their ability to break out of such traps and to achieve better solutions is based fundamentally on their initial solutions and neighborhood structures (Zhang *et al.* [29]). For tabu search, it has been shown that the quality of the final solution is strongly determined by the choice of initialization procedure (Jain *et al.* [13]). The major impact on solution quality produced by the initial solution suggests that further improvements may result from methods that are able to generate better initial solutions or to make better use of these initial solutions. Hence, to overcome the entrapment issue and to enhance the quality of the best solution, we propose a new diversification scheme that exploits the initial solutions and the neighborhood structures.

We identify our approach as Tabu Search Algorithm with multiple Restart (TSA-R) for total weighted tardiness job shop problems; our idea is based on a few observations. First, tabu search with solution recovery finds good solutions very effectively. Second, most local search procedures, such as tabu search, are unable to escape from strong local minima and have very poor diversification strategies. Third, for tabu search, the choice of initialization procedure has a strong influence on the quality of the final solutions. Nowicki and Smutnicki [21] discussed the efficacy of tabu search for the identification of a good solution. It is noted that tabu search is attracted to big valley areas and subsequently finds good solutions inside the big valleys. In a big valley landscape structure, local optima tend to exist close to one another in clusters, with each cluster centered on the global optimum that forms the valley structure. A landscape can be considered as a structure of the neighborhood generated by a heuristic operator that traverses the search space with some objective function. The big valley structure suggests that the determination of new starting points for search should be based on previous local optima rather than based on random points in the search space. This is because good candidate solutions are often found to be close to other good solutions.

Local search procedures based on neighborhood structures effectively apply intensification strategies, and are therefore currently among the best job shop scheduling techniques. They are able to achieve good solutions, but intensification is not enough. A strong diversification feature is also required to direct the search to other regions of the solution space so that the intensification process can evaluate these regions fully. Also by incorporating diversification, local minima can be transcended, improving the chances of finding the global optimum.

In TSA-R, the diversification strategy is incorporated into the initialization procedure, in order to avoid entrapment by strong local minima during early stages of the heuristic search. Our preliminary investigation found that it is usually much more difficult to escape from strong local minima at later stages than at the initial stage of TSAB. One would expect that our initial diversification might direct the search to explore more productive regions and one might hope that it would increase the chances of finding the global optimum. In TSA-R, we store a set of profitable initial solutions that are generated through NSP; the set has *MaxInitialSolns* members. This set also includes the initial solution that is generated by a selected heuristic method. A profitable solution is a schedule that improves the best solution at the NSP stage. Once *MaxInitialSoln*s solutions have been stored, each individual solution will then be subject to a varied set of parameter settings through the tabu search procedure described in section 5.1. The parameter settings are based on the results derived from the design of experiment (DOE) that applied the tabu search procedure to a set of benchmark problems. Only parameters that have a significant impact on the best solutions are considered. For each of the parameters, a range of values is applied. These values are determined to give optimal or near-optimal solutions in the DOE. The procedure of TSA-R is as follows:

Step 1. Generate and store the first *MaxInitialSolns* profitable solutions through NSP in a solution list in FIFO (first-in-first-out) order.

Step 2. Apply a specific set of parameter settings (based on DOE results) to TSAB for the first solution in the list, and store the best solution found so far.

Step 3. Repeat Step 2 for a different set of parameter settings until the settings are exhausted, and then remove the first solution from the list.

Step 4. Go to Step 2 for the next solution stored in the list until the list becomes empty.

Since our approach is a feature extension of TSAB, it should tend to improve solution quality but will definitely affect computing times.

6. **Computational results.** The proposed TSA-R was implemented in C# language, and the test runs were executed on an Intel (R) Core (TM) i7-3770 3.4 GHz processor and 16GB RAM. Two sets of benchmark instances have been used to evaluate the quality of the proposed TSA-R and other existing approaches. The first instance set comprised 22 10×10 ($m\times n$) problem instances, and the second instance set had 32 problems in a range of sizes from 10×20 to 10×50.

6.1. **Results for 10×10 problem instances.** We take the 22 cases of 10×10 problem instances used in Singer and Pinedo [25] (abz5, abz6, la16, la17, la18, la19, la20, la21, la22, la23, la24, mt10, orb1, orb2, orb3, orb4, orb5, orb6, orb7, orb8, orb9, orb10) from the OR library for comparison. We assign weights and due dates to those 22 cases, as was done in Singer and Pinedo [25]. The first 20% of the customers are very important. 60% of them are of average importance, and the remaining 20% are of little importance. Hence, we assign $w_1 = w_2 = 4$, $w_3 = w_4 = \ldots = w_8 = 2$, and $w_9 = w_{10} = 1$. The due date of job j is set to equal to the sum of all processing times p_{ij} of job j multiplied with a due date tightness factor f, *i.e.* $d_j = \lfloor f \times \sum_{i=1}^{m} p_{ij} \rfloor$. The comparisons involve three different levels of tardiness, denoted by $f = 1.3$, $f = 1.5$, and $f = 1.6$, as used in Singer and Pinedo [25]. The smaller the f value is, the tighter the generated due dates are. The optimal solutions for those three instances were obtained by Singer and Pinedo [25] with a branch and bound algorithm.

According to our extensive computational experimentation, the TSA-R related parameters were set to tabu list length $L_{maxt} = 12$, *CPType* = 0.1, *CPRandomType* = 0.1, *MaxElite* = 8, *MaxIters* = 15,000, *MaxNonImprovingIters* = 15,000, *RecoveryIter* = 50, *NonImprovingRecoveryIter* = 5, *MaxInitialSolns* = 20, and ATC scaling parameter $k_1 = 0.1$. *RecoveryIter* is associated with the maximum number of elite solution recoveries allowed. *NonImprovingRecoveryIter* applies to the recovery of elite solutions. Whenever a solution that is better than the best solution so far is found, the recovery iteration count will be recorded.

We compared TSA-R solutions to the optimal solutions provided by Singer and Pinedo [25], to SB-TS^1 ((3,2,2,3,2,2,1,1,1,1,)-RF with G/MAI) solutions, and to SB-TS^2 ((3,2,2,3,3,2,1,1,1,1,)-RF with G/MAI) solutions proposed by Bulbul [8]. We chose SB-TS to compare with since it is recently published, and it uses the tabu search algorithm as well. Bulbul [8] run SB-TS^1 and SB-TS^2 on a computer with a 2.4 GHz Intel Core 2 Quad Q6600 CPU with 3.25 GB of RAM. The average computation time for SB-TS^1 is 282.63, 155.68, and 229.28 seconds for f=1.3, 1.5, and 1.6, respectively. The average compuation time for SB-TS^2 is 356.19, 260.67, and 223.46 seconds for f=1.3, 1.5, and 1.6, respectively. Since we are using a faster computer, the proposed TSA-R was limited to 90 seconds. Table 2 gives the results of the 22 10×10 problem instances with three tardiness factors. We used * to represent the optimal solution that is achieved, and used bold value to represent the optimal solution that is improved.

When due dates are tight ($f = 1.3$), the proposed TSA-R performs the best. The TSA-R on average deviates 1.24% from the optimal solutions, and there are 15 out of 22 instances in which it solves the instances to optimality or are improved. The SB-TS^1 on average deviates 1.47% from the optimal solutions, and there are 12 out of 22 instances in which it solves the instances to optimality or is improved. The SB-TS^2 on average deviates 1.30% from the optimal solutions, and there are 12 out of 22 instances in which it solves the instances to optimality or is improved.

Similarly, when the due date factor is $f = 1.5$, the proposed TSA-R performs the best. The TSA-R on average deviates 2.21% from the optimal solutions, and there are 16 out of 22 instances in which it solves the instances to optimality or is improved. The SB-TS^1 on average deviates 3.88% from the optimal solutions, and there are 15 out of 22 instances in which it solves the instances to optimality or improved. The SB-TS^2 on average deviates 3.03% from the optimal solutions, and there are 17 out of 22 instances in which it solves the instances to optimality

TABLE 2. Results for 10×10 instances (TSA-R limited to 90 seconds)

	f=1.3				f=1.5				f=1.6			
Instances	Opt.	SB-TS^1	SB-TS^2	TSA-R	Opt.	SB-TS^1	SB-TS^2	TSA-R	Opt.	SB-TS^1	SB-TS^2	TSA-R
abz5	1405	1462	1462	**1403**	69	70	*	*	0	*	*	*
abz6	436	*	*	*	0	*	*	*	0	*	*	*
la16	1170	**1169**	**1169**	**1169**	166	*	*	*	0	*	*	*
la17	900	**899**	**899**	**899**	260	*	*	*	65	*	*	*
la18	929	*	*	*	34	*	*	36	0	*	*	*
la19	948	955	955	*	21	23	23	*	0	*	*	*
la20	809	**805**	**805**	819	0	1	1	*	0	*	*	*
la21	464	**463**	**463**	**463**	0	*	*	*	0	*	*	*
la22	1068	1084	1084	1107	196	*	*	*	0	*	*	*
la23	837	877	877	873	2	*	*	*	0	*	*	*
la24	835	*	*	*	82	*	*	88	0	*	*	*
mt10	1368	**1363**	**1363**	**1363**	394	*	*	*	141	155	155	*
orb1	2568	2630	2630	*	1098	1202	1202	1124	566	776	619	*
orb2	1412	**1408**	**1408**	**1408**	292	322	*	*	44	52	52	*
orb3	2113	2115	2115	2186	918	952	928	947	422	461	461	426
orb4	1623	1652	1652	1645	358	*	*	*	66	*	*	*
orb5	1593	*	*	1667	405	*	*	472	163	181	181	*
orb6	1792	**1790**	**1790**	**1790**	426	*	*	*	31	*	*	**28**
orb7	590	616	616	*	50	*	*	*	0	*	*	*
orb8	2429	2503	2453	2541	1023	*	*	1035	621	672	672	675
orb9	1316	*	*	*	297	*	*	*	66	*	*	*
orb10	1679	1801	1801	*	346	424	424	*	76	84	78	*

or is improved. Finally, when the due dates are loose ($f = 1.6$), the proposed TSA-R performs the best. The TSA-R on average deviates 2.43% from the optimal solutions, and there are 20 out of 22 instances in which it solves the instances to optimality or is improved. The SB-TS^1 on average deviates 15.39% from the optimal solutions, and there are 15 out of 22 instances in which it solves the instances to optimality or is improved. The SB-TS^2 on average deviates 8.18% from the optimal solutions, and there are 15 out of 22 instances in which it solves the instances to optimality or is improved. Overall, the TSA-R outperforms the SB-TS^1 and SB-TS^2 in terms of total weighted tardiness and the number of optimal values achieved.

Next, we compare TSA-R with MDL proposed by Mati *et al.* [18]. MDL is a three-step based approach. The approach is a local search method that uses a disjunctive graph model and neighborhoods generated by swapping critical arcs. Mati *et al.* [18] limited MDL to evaluate at most 200,000 solutions per run, and showed that MDL was outperforming the genetic algorithm of Zhou *et al.* [35]. Here, we also limited TSA-R to evaluate 200,000 solutions at most. In TSA-R, each move (refer to Fig.4) generates a new solution. Hence, we limited TSA-R to at 200,000 moves at most. The results are shown in Table 3. Mati *et al.* [18] executed 10 independent runs per instance. In Table 3, columns mean and best represent the mean and best values obtained by MDL, respectively. For the case $f = 1.3$, the mean values of MDL, the best values of MDL, and the TSA-R on average deviates 20.36%, 7.58%, and 0.24% from the optimal solutions, respectively. For the case $f = 1.5$, the mean values of MDL, the best values of MDL, and the TSA-R on average deviates 63.09%, 21.69%, and 1.58% from the optimal solutions, respectively. For the case $f = 1.5$, the mean values of MDL, the best values of MDL, and the TSA-R on average deviates 85.10%, 37.86%, and 2.34% from the optimal solutions, respectively. Moreover, out of 66 instances, the mean value of MDL found optimal solutions in 10 instances and 0 solutions better than optimal solutions. The best values of MDL found 25 optimal solutions and 6 solutions better than optimal solutions. The TSA-R found 48 optimal solutions and 10 solutions better than optimal solutions. Overall, the TSA-R outperforms the MDL in terms of total weighted tardiness and the number of optimal values achieved when both were limited to 200,000 evaluations.

In Tables 2 and 3, there are some instances of our proposed TSA-R, SB-TS[1] , SB-TS[2] and MDL that obtained better solutions than those reported by Pinedo and Singer [23]. This might be because the branch-and-bound algorithm was either stopped prematurely or because the due dates were inadvertently made too tight (Bulbul, [8]).

TABLE 3. Results for 10×10 instances (TSA-R limited to 200,000 evaluations

	f=1.3				f=1.5				f=1.6			
Instances	Opt.	MDL		TSA	Opt.	MDL		TSA	Opt.	MDL		TSA
		mean	best	-R		mean	best	-R		mean	best	-R
abz5	1405	1521	1443	**1403**	69	209	100	*	0	*	*	*
abz6	436	568	*	*	0	*	*	*	0	*	*	*
la16	1170	1343	1223	**1169**	166	366	212	*	0	*	*	*
la17	900	989	**899**	**899**	260	344	*	*	65	115	*	*
la18	929	1248	1132	*	34	285	124	*	0	*	*	*
la19	948	1127	955	*	21	77	*	*	0	*	*	*
la20	809	937	**805**	**805**	0	23	*	*	0	*	*	*
la21	464	477	**463**	**463**	0	11	4	*	0	*	*	*
la22	1068	1220	1154	1082	196	359	281	*	0	42	*	*
la23	837	1079	873	**835**	2	67	*	*	0	*	*	*
la24	835	1082	*	*	82	111	94	88	0	*	*	*
mt10	1368	1802	1685	**1363**	394	481	*	*	141	214	176	*
orb1	2568	3252	2677	*	1098	1721	1334	*	566	964	780	*
orb2	1412	1797	1541	**1408**	292	444	403	*	44	148	56	*
orb3	2113	2451	**2111**	2115	918	1288	1019	947	422	744	577	461
orb4	1623	1837	1789	*	358	540	*	*	66	106	*	*
orb5	1593	2104	1994	1667	405	771	*	472	163	299	193	180
orb6	1792	1980	**1790**	**1790**	426	643	*	*	31	78	**28**	**28**
orb7	590	642	612	*	50	120	103	*	0	2	*	*
orb8	2429	2973	2828	*	1023	1642	1477	*	621	1056	864	*
orb9	1316	1641	*	*	297	390	352	*	66	159	140	*
orb10	1679	1974	1868	*	346	606	464	*	76	258	172	*

6.2. **Other problem sizes.** We next tested the TSA-R on other benchmark problem instances taken from the OR-Library. These 32 instances belong to the swv, la, abz, and yn classes, which include operations from 200 to 500. Since these 32 instances are originally served as a benchmark for the $J_m||C_{max}$ problem, they do not include weights and due dates. The weights and due dates were generated by the same approach that Singer and Pinedo [25] described above. Similarly, three values of tightness factors f = 1.3, 1.5, and 1.6 are considered. The proposed TSA-R was limited to 200 seconds. Since those large scale problem instances do not have optimal solutions by which they can be compared, we compare the TSA-R with the best solutions obtained by WEDD, MDD, ATC, and SPT. The results are shown in Table 4. Under best heuristic column, we used superscript a, b, and c to indicate which heuristic obtained the best solution. For larger scale problem instances, there are 67 out of 96 instances, in which the WEDD obtained the best solution. There are 23 out of 96 instances,in which the MDD obtained the best solution, and there are only 6 out of 96 instances, in which the ATC obtained the best solution. The best solution identified by the three dispatching rules on average deviates 15.82%, 17.61%, and 17.87% from the TSA-R for cases f=1.3, f=1.5, and f=1.6, respectively.

TABLE 4. Results of TSA-R for larger problem sizes (TSA-R limited to 200 seconds)

			f=1.3		*f=1.5*		*f=1.6*	
Instance	m	n	Best Heuristic	TSA -R	Best Heuristic	TSA -R	Best Heuristic	TSA -R
swv01	10	20	28833[a]	21575	24589[a]	17585	22712[a]	13458
swv02	10	20	25716[a]	14371	21742[a]	12083	19868[a]	10915
swv03	10	20	30878[b]	16361	24853[b]	16690	23576[b]	11152
swv04	10	20	22964[a]	15431	18511[a]	11068	16399[a]	9642
swv05	10	20	22284[a]	16596	18211[a]	11640	16535[a]	11417
la31	10	30	51598[a]	44244	44721[a]	36223	41305[a]	32852
la 32	10	30	59998[b]	53708	49963[b]	44861	48262[b]	41049
la 33	10	30	56281[a]	46762	49682[a]	40478	46400[a]	37729
la 34	10	30	57508[b]	48768	50776[b]	41935	49724[b]	41815
la 35	10	30	54310[c]	46985	47402[c]	39799	43968[c]	35400
abz7	15	20	8410[b]	4279	5182[b]	2036	3649[b]	750
abz 8	15	20	7789[b]	3586	5220[a]	1500	3143[b]	654
abz 9	15	20	9291[a]	3487	6186[a]	1746	4689[a]	867
swv06	15	20	34315[b]	26316	27588[b]	20269	24595[b]	19041
swv07	15	20	31868[b]	23457	27743[b]	10428	24819[b]	9969
swv08	15	20	36778[a]	27682	30364[a]	17596	27179[a]	15490
swv09	15	20	29366[a]	20774	23083[a]	12580	20173[a]	14613
swv10	15	20	33592[a]	25261	26523[a]	20111	23028[a]	16556
yn1	20	20	5832[a]	1449	1616[a]	6	515[a]	0
yn2	20	20	7377[a]	3458	2814[a]	466	1376[a]	0
yn3	20	20	6890[b]	2597	2208[b]	12	802[b]	0
yn4	20	20	9436[a]	5005	4669[a]	1003	2490[a]	0
swv11	10	50	194893[c]	180619	183442[c]	168857	173196[c]	157739
swv12	10	50	171325[a]	156748	159985[a]	145408	154342[a]	139369
swv13	10	50	168299[a]	144871	157065[a]	132883	151617[a]	126592
swv14	10	50	170230[a]	148223	159455[a]	141702	154094[a]	133625
swv15	10	50	180218[a]	160586	169112[a]	151103	163620[a]	145882
swv16	10	50	164891[a]	155063	153658[a]	140779	148056[a]	136067
swv17	10	50	169785[a]	159552	158851[a]	145649	153414[a]	140014
swv18	10	50	158288[a]	141652	147191[a]	130355	141665[a]	125643
swv19	10	50	173343[a]	159095	162312[a]	147666	156827[a]	142151
swv20	10	50	162243[a]	145946	151205[a]	134615	145718[a]	132985

a:WEDD b:MDD c: ATC

7. **Conclusions and future works.** In this research, we have presented a tabu search algorithm with a restart (TSA-R) for minimizing total weighted tardiness of job shop scheduling problems. We tested two sets of benchmark problems to evaluate the quality of the proposed TSA-R and other existing approaches. The first problem set comprised 22 10×10 ($m \times n$) problem instances. When the TSA-R computation time was limited to 90 seconds, the TSA-R outperformed existing heuristic SB-TS in terms of total weighted tardiness and the number of instances were solved to optimality. When the TSA-R was limited to 200,000 evalutations, the TSA-R outperformed existing heuristic MDL in terms of the total weighted tardiness, and the number of instances solved to optimality. The second set contained 32 problems, and covered a range of sizes from 10×20 to 10×50. Computational results have shown that the TSA-R deviates the best heuristic by 15.82%, 17.61%, and 17.87% for three different due date tightness factors *f*.

In future research, we would like to further examine the experimental study of the TSA-R approach, fine-tuning each problem instance with different parameter settings. In this way, we should be able to find even better solutions. Another useful research direction could be automatic, dynamic, adaptive parameter changes within TSA-R. Automatic parameter adjustment could overcome the inconvenience of manually fine-tuning a large number of combinations for each instance.

REFERENCES

[1] E. J. Anderson and J. C. Nyirenda, Two new rules to minimize tardiness in a job shop, *International Journal of Production Research*, **28** (1990), 2277–2292.

[2] V. A. Armentano and C. R. Scrich, Tabu search for minimizing total tardiness in a job shop, *International Journal of Production Research*, **63** (2000), 131–140.

[3] M. Asano and H. Ohta, A heuristic for job shop scheduling to minimize total weighted tardiness, *Computers and Industrial Engineering*, **42** (2002), 137–147.

[4] K. R. Baker, Sequencing rules and due date assignments in a job shop, *Management Science*, **30** (1984), 1093–1104.

[5] K. R. Baker and J. J. Kanet, Job shop scheduling with modified due dates, *Journal of Operations Management*, **4** (1983), 11–22.

[6] E. Balas, Machine sequencing via disjunctive graph: an implicit enumeration algorithm, *Operations Research*, **17** (1969), 941–957.

[7] K. M. J. Bontridder, Minimizing total weighted tardiness in a generalized job shop, *Journal of Scheduling*, **8** (2005), 479–496.

[8] K. Bulbul, A hybrid shifting bottleneck-tabu search heuristic for the job shop total weighted tardiness problem, *Computers and Operations Research*, **38** (2011), 967–983.

[9] G. Calleja and R. Pastor, A dispatching algorithm for flexible job-shop scheduling with transfer batches: an industrial application, *Production Planning and Control*, **25** (2014), 93–109.

[10] I. Essafi, Y. Mati and S. Dauzere-Peres, A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem, *Computers and Operations Research*, **35** (2008), 2599–2616.

[11] R. L. Graham, E. L. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, *Annals of Discrete Mathematics*, **5** (1979), 287–326.

[12] Z. He, T. Yang and D. E. Deal, A multiple-pass heuristic rule for job shop scheduling with due dates, *International Journal of Production Research*, **31** (1993), 2677–2692.

[13] A. S. Jain, B. Rangaswamy and S. Meeran, New and "stronger" job-shop neighborhoods: a focus on the method of Nowicki and Smutnicki (1996), *Journal of Heuristics*, **6** (2000), 457–480.

[14] J. J. Kanet and J. C. Hayya, Priority dispatching with operation due dates in a job shop, *Journal of Operations Management*, **2** (1982), 167–175.

[15] S. Kreipl, A large step random walk for minimizing total weighted tardiness in a job shop, *Journal of Scheduling*, **3** (2000), 125–138.

[16] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys, Sequencing and scheduling: Algorithms and complexity, in *Handbooks in Operations Research and Management Science* (eds. S. C. Graves, A. H. G. Rinnooy Kan and P. H. Zipkin), **4** (1993), 445–522.

[17] H. L. Lu, G. Q. Huang and H. D. Yang, Integrating order review/release and dispatching rules for assembly job shop scheduling using a simulation approach, *International Journal of Production Research*, **49** (2011), 647–669.

[18] Y. Mati, S. Dauzere-Peres and C. Lahlou, A general approach for optimizing regular criteria in the job-shop scheduling problem, *European Journal of Operational Research*, **212** (2011), 33–42.

[19] D. C. Mattfeld and C. Bierwirth, An efficient genetic algorithm for job shop scheduling with tardiness objectives, *European Journal of Operational Research*, **155** (2004), 616–630.

[20] E. Nowicki and C. Smutnicki, A fast taboo search algorithm for the job shop problem, *Management Science*, **42** (1996), 797–813.

[21] E. Nowicki and C. Smutnicki, An advanced tabu search algorithm for the job shop problem, *Journal of Scheduling*, **8** (2005), 145–159.

[22] S. Nguyen, M. Zhang, J. Mark and KC. Tan, A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem, *IEEE Transactions on Evolutionary Computation*, **17** (2013), 621–639.

[23] M. Pinedo and M. Singer, A shifting bottleneck heuristic for minimizing the total weighted tardiness in job shop, *Naval Research Logistics*, **46** (1999), 1–17.

[24] N. Raman and F. B. Talbot, The job shop tardiness problem: A decomposition approach, *European Journal of Operational Research*, **69** (1993), 187–199.

[25] M. Singer and M. Pinedo, A computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops, *IIE Transactions*, **30** (1998), 109–118.

[26] P. J. M. Van Laarhoven, E. H. L. Aarts and J. K. Lenstra, Job shop scheduling by simulated annealing, *Operations Research*, **40** (1992), 113–125.
[27] A. P. J. Vepsalainen and T. E. Morton, Priority rules for job shops with weighted tardiness costs, *Management Science*, **33** (1987), 1035–1047.
[28] T. Yang, Z. He and K. K. Cho, An effective heuristic method for generalized job shop scheduling with due dates, *Computers and Industrial Engineering*, **26** (1994), 647–660.
[29] C. Zhang, X. Shao, Y. Rao and H. Qiu, Some new results on tabu search algorithm applied to the job-shop scheduling problem, Tabu Search, Wassim Jaziri (Ed.), InTech, Available from: http://www.intechopen.com/articles/show/title/some_new_results_on_tabu_search_algorithm_applied_to_the_job-shop_scheduling_problem (2008).
[30] R. Zhang, A genetic local search algorithm based on insertion neighborhood for the job shop scheduling problem, *Advances in Information Sciences and Services*, **3** (2011), 117–125.
[31] R. Zhang and C. Wu, A divide-and-conquer strategy with particle swarm optimization for the job shop scheduling problem, *Engineering Optimization*, **42** (2010a), 641–670.
[32] R. Zhang and C. Wu, A hybrid immune simulated annealing algorithm for the job shop scheduling problem, *Applied Soft Computing*, **10** (2010b), 79–89.
[33] R. Zhang and C. Wu, A simulated annealing algorithm based on block properties for the job shop scheduling problem with total weighted tardiness objective, *Computers and Operations Research*, **38** (2011), 854–867.
[34] R. Zhang, S. Song and C. Wu, A hybrid artificial bee colony algorithm for the job shop scheduling problem, *International Journal of Production Economics*, **141** (2013), 167–178.
[35] H. Zhou, W. Cheung and L. C. Leung, Minimizing weighted tardiness of job-shop scheduling using a hybrid genetic algorithm, *European Journal of Operational Research*, **194** (2009), 637–649.

Received October 2014; 1st revision March 2015; 2nd revision April 2015.

E-mail address: yklin@mail.fcu.edu.tw
E-mail address: cschong@SIMTech.a-star.edu.sg