Check for updates

INTECH
open science | open minds

OPEN ACCESS
ARTICLE

International Journal of Advanced Robotic Systems

A Simulated Annealing-based Heuristic Algorithm for Job Shop Scheduling to Minimize Lateness

Regular Paper

Rui Zhang[1,*]

1 School of Economics and Management, Nanchang University, Nanchang, PR China
* Corresponding author E-mail: r.zhang@ymail.com

Received 3 Nov 2012; Accepted 24 Jan 2013

DOI: 10.5772/55956

© 2013 Zhang; licensee InTech. This is an open access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Abstract A decomposition-based optimization algorithm is proposed for solving large job shop scheduling problems with the objective of minimizing the maximum lateness. First, we use the constraint propagation theory to derive the orientation of a portion of disjunctive arcs. Then we use a simulated annealing algorithm to find a decomposition policy which satisfies the maximum number of oriented disjunctive arcs. Subsequently, each subproblem (corresponding to a subset of operations as determined by the decomposition policy) is successively solved with a simulated annealing algorithm, which leads to a feasible solution to the original job shop scheduling problem. Computational experiments are carried out for adapted benchmark problems, and the results show the proposed algorithm is effective and efficient in terms of solution quality and time performance.

Keywords Job Shop Scheduling, Simulated Annealing, Optimization Algorithm, Lateness

1. Introduction

Scheduling has been an important research field in robotics and computer-integrated manufacturing [1, 2]. The job shop scheduling problem (JSSP) is one of the most frequently adopted models in the area of scheduling research [3–6]. However, most variants of JSSP are $\mathcal{NP}$-hard in the strong sense and thus defy ordinary solution methods. Enumerative approaches, such as the branch-and-bound algorithm [7], can only conquer small-scale problem instances. The most common heuristic methods devised in the early days include dispatching rules [8–10] and shifting bottleneck [11–13]. In recent years, meta-heuristic algorithms, such as simulated annealing (SA) [14], genetic algorithms (GA) [15], scatter search (SS) [16], tabu search (TS) [17] and particle swarm optimization (PSO) [18], have clearly become the research focus in practical optimization methods for solving ordinary-scale JSSPs.

However, the size of JSSPs in practical manufacturing environments is much larger than in theoretical research. In this case, the solution space is too large to be searched effectively by simple meta-heuristics. To address this difficulty, several decomposition-based algorithms have been proposed [19–23], which decompose the original large-scale problem into a series of smaller problems (subproblems) and finally obtain a solution to the original problem after these subproblems are solved respectively. But there are some drawbacks with these existing

approaches. The decomposition algorithm proposed in [19, 22] has exponential complexity and thus cannot be directly applied to large-scale problems; the algorithm in [20] encounters deterioration in solution quality because it adds extra constraints to each subproblem; the method presented in [21] does not guarantee the feasibility of subproblem solutions for the original problem and relies on a heuristic coordination algorithm to construct a feasible solution for the original problem.

In this paper, we propose a decomposition-optimization algorithm based on simulated annealing (DOASA) to solve large-scale JSSPs with the objective of minimizing maximum lateness. In DOASA, the original problem will be divided into several subproblems, each of which corresponds to a subset of operations. The subproblems are first determined and then successively solved by two optimization processes, both of which are implemented with simulated annealing in order to save computational time. Before decomposition, the constraint propagation technique is applied as a preprocessor on the original problem so that some disjunctive arcs can be resolved. This information provides guidance for the determination of decomposition policy and subproblems.

The paper is organized as follows. The discussed job shop scheduling problem is formulated in Section 2. Section 3 introduces the important tools that are used by our algorithm. Section 4 describes the DOASA in detail. The computational results are provided in Section 5. Finally, some conclusions are given in Section 6.

2. Problem formulation

In a JSSP instance, a set of n jobs $\{J_j\}_{j=1}^{n}$ are to be processed on a set of m machines $\{M_k\}_{k=1}^{m}$ such that each machine can process only one job at a time, and each job may be processed by only one machine at a time. Each job has a fixed processing route which traverses all the machines in a predetermined order. A preset due date is given for each job. Since due date related criteria are of much greater concern in practical scheduling, maximum lateness is adopted as the optimization objective.

JSSP can be described by a disjunctive graph $G(N, C, D)$ [23]. $N = O \cup \{0, *\}$ is the set of nodes where $O = \{1, 2, \ldots, n \times m\}$ are associated with the $n \times m$ operations of the JSSP instance. 0 and $*$ denote the dummy source node and the dummy sink node, respectively. C is the set of conjunctive arcs which connect successive operations of the same job. Thus, C models the technological constraints in the JSSP instance. $D = \bigcup_{k=1}^{m} D_k$ is the set of disjunctive arcs if D_k is used to denote the disjunctive arcs that correspond to the operations on machine k. D_k contains the arcs that connect each pair of operations to be processed by machine k and require that these operations cannot be processed simultaneously. Therefore, the directions of the disjunctive arcs are yet to be determined. Finding a feasible schedule for JSSP is equivalent to orienting all the disjunctive arcs without causing any directed cycles in the resulting graph. The weight of each arc equals the processing time of the operation that is associated with the tail of the arc.

The discussed JSSP can be formulated as follows:

$$\begin{aligned} \min \ & L_{\max} = \max_{i \in F(O)} \{t_i + p_i - d_i\} \\ \text{s.t.} \ & \\ & t_i + p_i \le t_j, \quad (i,j) \in C, \\ & t_i + p_i \le t_j \text{ or } t_j + p_j \le t_i, \quad (i,j) \in D, \\ & t_i \ge 0, \quad i \in O. \end{aligned}$$

In this formulation, $F(O)$ is the set of the last operations of each job; d_i is the due date of the job which operation i belongs to; p_i and t_i are the processing time and the starting time of operation i, respectively. The lateness of job j is defined as $L_j = C_j - d_j$ where C_j equals the completion time of the last operation of job j. The scheduling objective considered in this paper is to determine the processing sequence of operations on each machine such that the maximum lateness (i.e. $L_{\max} = \max_{j=1}^{n}\{L_j\}$) is minimized. The problem is noted as $Jm||L_{\max}$ in accordance with the three-field notation.

3. Theoretical background

3.1. Simulated annealing (SA)

The SA algorithm is commonly used for solving combinatorial optimization problems. It starts from an initial basic solution x_0 and randomly generates a neighbour solution x' from its neighbourhood. x' is accepted as a new basic solution with probability P, which is calculated as

$$P = \begin{cases} 1, & \text{if } \Delta L_{\max} < 0 \\ \exp\{-\Delta L_{\max}/T\}, & \text{otherwise} \end{cases}$$

where $\Delta L_{\max} = L_{\max}(x') - L_{\max}(x_0)$ is the difference in objective value between the two solutions, and T is the temperature which decreases with iterations. The searching process keeps generating a neighbour solution and accepting it with the above probability until some stopping criterion is met. The framework of the standard SA algorithm can be described as follows.

Step 1: Produce an initial solution x_0. Set $x^* = x_0$.

Step 2: For $iter = 1, 2, \ldots, I_{\max}$

(2.1) Randomly generate a neighbour solution x' from the neighbourhood of x_0.

(2.2) If $L_{\max}(x') < L_{\max}(x^*)$, set $x^* = x'$.

(2.3) Set $x_0 = x'$ with probability $P = \min\{1, \exp\{(L_{\max}(x_0) - L_{\max}(x'))/T\}\}$.

(2.4) Reduce the temperature by setting $T := T \times \lambda$, where $\lambda \in (0, 1)$ is the cooling rate.

Step 3: Return x^*.

3.2. Constraint propagation (CP)

In this part, we will introduce the basic application of CP theory to job shop scheduling.

We use $\mathcal{P} = (O, C, D, P)$ to denote a specific JSSP instance. As introduced above, $O = \{1, \ldots, n \times m\}$ is the set of operations; C is the set of conjunctive arcs and $(i, j) \in C$ indicates operation i must be processed before operation

j; D is the set of disjunctive arcs and $(i,j) \in D$ indicates operation i cannot overlap with operation j in processing; $P = \{p_i | i \in O\}$ is the set of operation processing times.

In order to apply CP to JSSP, it is required to first give an upper bound UB for the optimal $L_{\max}$ of the instance. Therefore, if we use δ_i to denote the feasible domain of starting times of each operation $i \in O$, we must have $\delta_i \subseteq [0, UB + d_i - p_i]$. Normally, the tighter the upper bound, the more information will be derived by CP.

The earliest starting time and the latest starting time of operation i are respectively defined as $est_i = \min \delta_i$, $lst_i = \max \delta_i$. Thus, δ_i can also be expressed as $\delta_i = [est_i, lst_i] = \{est_i, est_i + 1, \ldots, lst_i - 1, lst_i\}$. Meanwhile, we define the earliest completion time and the latest completion time of operation i respectively as $ect_i = est_i + p_i$ and $lct_i = lst_i + p_i$.

For a subset of operations $A \subseteq O$ (which need to be processed on the same machine), we define the following functions: $P(A) = \sum_{i \in A} p_i$, $EST_{\min}(A) = \min_{i \in A} est_i$ and $LCT_{\max}(A) = \max_{i \in A} lct_i$.

In the following, some of the most important principles for applying CP to JSSP are presented [24].

Theorem 1 (Conjunctive consistency test)**.** *If $i, j \in O$ and $(i,j) \in C$, then the following domain reduction rule is effective:*

$$est_j := \max\{est_j, est_i + p_i\},$$
$$lst_i := \min\{lst_i, lst_j - p_i\}.$$

Clearly, the time complexity of revising the earliest/latest starting times is $O((nm)^2)$.

Theorem 2 (Input/output sequence consistency test)**.** *Assume $A \subsetneq O$ and $i \notin A$, if the following "input condition" is satisfied:*

$$\max_{u \in A, v \in A \cup \{i\}, u \neq v} \{lct_v - est_u\} < P(A \cup \{i\}),$$

then operation i must be completed before every operation in A. Similarly, if the following "output condition" is satisfied:

$$\max_{u \in A \cup \{i\}, v \in A, u \neq v} \{lct_v - est_u\} < P(A \cup \{i\}),$$

then operation i must be processed after every operation in A.

When $|A| = 1$, the above theorem can be stated as: if $i \neq j$ and $lct_j - est_i < p_i + p_j$, then the disjunctive arc direction $j \to i$ can be fixed. Such a "pair-wise test" can be performed in $O((nm)^2)$ time.

Theorem 3 (Output domain consistency test)**.** *If there exist $A \subsetneq O$ and $i \notin A$ which satisfy the output condition, then the earliest starting time of operation i can be adjusted as*

$$est_i := \max\{est_i, EST_{\min}(A) + P(A)\}.$$

Naturally, a dual conclusion exists for describing the input condition.

Theorem 4 (Input/output negation sequence consistency test)**.** *Assume $A \subsetneq O$ and $i \notin A$, if the following "input negation condition" is satisfied:*

$$LCT_{\max}(A) - est_i < P(A \cup \{i\}),$$

then operation i cannot be completed before every operation in A. Similarly, if the following "output negation condition" is satisfied:

$$lct_i - EST_{\min}(A) < P(A \cup \{i\}),$$

then operation i cannot be processed after every operation in A.

Theorem 5 (Input negation domain consistency test)**.** *If there exist $A \subsetneq O$ and $i \notin A$ which satisfy the input negation condition, then the earliest starting time of operation i can be adjusted as*

$$est_i := \max\left\{est_i, \min_{u \in A}\{ect_u\}\right\}.$$

Naturally, a dual conclusion exists for describing the output negation condition.

Besides the listed theorems, there are some others, such as the input-or-output consistency test. In the subsequent part of the paper, the application of CP to JSSP is regarded as an independent module, the procedure of which can be found in [25].

4. The algorithm

4.1. The decomposition method

The decomposition scheme used in this paper can be intuitively illustrated using a disjunctive graph like Figure 1, in which nine operations of three jobs are to be processed on three machines. According to the conjunctive arcs, operations 1-3 belong to job 1, operations 4-6 belong to job 2 and operations 7-9 belong to job 3. According to the disjunctive arcs, operations 1, 6 and 8 are to be processed by machine 1, operations 2, 4 and 7 are to be processed by machine 2, and operations 3, 5 and 9 are to be processed by machine 3. The dividing curves representing a decomposition policy divide the operation set into three sequential subsets: I, II and III, each of which corresponds to a subproblem.

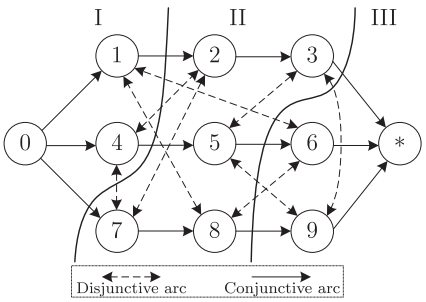

Figure 1. An illustration of the decomposition method

To be precise, the decomposition method has the following properties: (1) If the original operation set is divided into p subsets $\{O_l\}_{l=1}^{p}$, then $\bigcup_{l=1}^{p} O_l = O$, and $\forall l \neq l'$, $O_l \cap O_{l'} = \emptyset$; (2) For any two operations i and j, if $(i,j) \in C$ and $i \in O_{l_i}$, $j \in O_{l_j}$, then $l_i \leq l_j$.

4.2. Using CP as a preprocessor for JSSP

From Section 3.2, we see that CP can derive the correct direction of some disjunctive arcs by alternating sequence consistency tests and domain consistency tests. The following algorithm is designed by combining CP with local search in an iterative fashion so that a tight upper bound can be used to orient a sufficient number of disjunctive arcs.

Input: The problem instance $\mathcal{P} = (O, C, D, P)$.

Step 1: Construct the initial solution S_1 based on the EDD (earliest due date) dispatching rule. Let $UB = L_{\max}(S_1)$, where $L_{\max}(\cdot)$ denotes the objective value of a solution. Let the initial set of directed disjunctive arcs $\hat{D} = \emptyset$ and $k = 1$.

Step 2: Perform a stochastic local search starting from S_k (employing w iterations) in order to find a new best-so-far solution S^* and update the upper bound $UB = L_{\max}(S^*)$. If no improvement is found within the w iterations, then output $\hat{D}$.

Step 3: Add a set of new constraints "$\forall i \in O$, $lct_i := \min\{lct_i, UB - \Delta\}$" and then call the CP module to derive as much information as possible. The disjunctive arc directions newly obtained in this step are denoted as D'. If $\delta_i = \emptyset$ for some operation or $C \cup \hat{D} \cup D'$ contains cycles, output $\hat{D}$. Otherwise, let $\hat{D} := \hat{D} \cup D'$ and $k := k + 1$.

Step 4: Taking $C \cup \hat{D}$ as the precedence constraints, construct a new solution S_k based on the EDD rule. If $L_{\max}(S_k) < UB$, let $UB = L_{\max}(S_k)$. Go back to Step 2.

The above algorithm can be used as a preprocessor which is applied to a JSSP instance before we start to solve it. The two parameters involved in this procedure are w (the number of iterations in each run of local search) and Δ (the step size when lowering the upper bound). The resolved disjunctive arcs $\hat{D}$ will be used to determine a decomposition policy for the JSSP instance.

4.3. Finding a decomposition policy

A decomposition policy determines the structures of all the subproblems at a time and thus it directly affects the subsequent optimization process. In this paper, a simulated annealing (SA) approach is designed to search for a satisfactory decomposition policy based on the previously derived disjunctive arc directions.

SA searches among all possible decomposition policies and outputs a promising policy, which divides the operation set into p successive subsets by satisfying the maximum number of derived disjunctive arcs. The input parameters include the desired number of operations in each produced subset, i.e., $\{\alpha_l\}_{l=1}^{p}$.

A directed disjunctive arc (i, j) is "satisfied" if the subset containing operation i is not preceded by the subset containing operation j. With respect to the previous example (see Figure 2 here), if the disjunctive arcs $(1, 8)$ and $(6, 8)$ have been oriented by CP, then the illustrated decomposition policy satisfies $(1, 8)$ (because operation 1 is in subset I and operation 8 is in subset II), but does not satisfy $(6, 8)$ (because subset II, which includes operation 8, precedes subset III which includes operation 6). To be precise, suppose $(i, j) \in \hat{D}$ and the considered decomposition policy (dividing the original operation set into p successive subsets $\{O_l\}_{l=1}^{p}$) indicates $i \in O_{l_i}$, $j \in O_{l_j}$, then (i, j) is satisfied only if $l_i \leq l_j$.

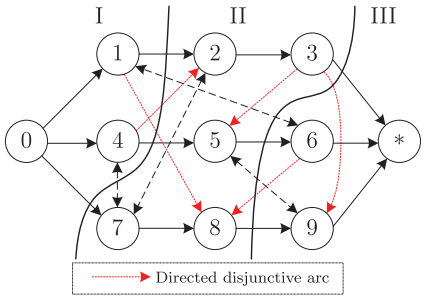

Figure 2. An illustration of satisfied and dissatisfied arcs

4.3.1. Encoding

A matrix $\mathbf{B} = [b_{j,l}]_{n \times p}$ is used to encode a solution, where the element $b_{j,l}$ denotes the number of operations that belong to job j and have been assigned to subset O_l. According to this definition, $\mathbf{B}$ has to satisfy:

$$\sum_{l=1}^{p} b_{j,l} = m,\ j = 1, \ldots, n,$$
$$\sum_{j=1}^{n} b_{j,l} = \alpha_l,\ l = 1, \ldots, p.$$

In the above equations, m is exactly the number of operations included by each job. Besides, since the operations of each job are decomposed according to their technological order, a matrix in the form of $\mathbf{B}$ is sufficient for encoding a solution in the SA.

For example, the decomposition policy shown in Figure 2 should be encoded as

$$\mathbf{B}_0 = \begin{bmatrix} 1 & 2 & 0 \\ 1 & 1 & 1 \\ 0 & 2 & 1 \end{bmatrix},$$

where the first row means the 1st operation of job 1 is assigned to the subset I, the next two operations of job 1 are assigned to subset II, while no operation of job 1 is assigned to subset III.

4.3.2. Initialization

A heuristic rule similar to EDD is used to produce the initial solution. For each operation i in O, we define the modified due date h_i as:

$$h_i = \begin{cases} d_i, & \text{if } \{j : (i, j) \in C \cup \hat{D}\} = \emptyset \\ \min\limits_{(i,j) \in C \cup \hat{D}} \{h_j - p_j\}, & \text{otherwise} \end{cases}$$

where d_i is the due date of the job to which operation i belongs. Just like d_i, the newly defined h_i also describes

the urgency level of operation i, but in a more strict manner than the original due date.

Then, all the operations are sorted in a non-decreasing order of h_i, yielding an operation sequence $[i_1, i_2, \ldots, i_{nm}]$. After that, the p subsets are successively assigned with operations in the order specified by this sequence.

According to the encoding scheme, the value of $b_{j,l}$ (the number of job j's operations that have been assigned to subset O_l) can now be acquired, and thus, the encoding matrix $\mathbf{B}^{(1)}$ for this initial decomposition policy is obtained.

4.3.3. New solution generation

In order to generate a new solution from the current solution, the algorithm randomly selects an element $b_{j,l}$ from $\mathbf{B}$ and sets $b_{j,l} \leftarrow b_{j,l} + \xi$ where ξ is a two-value random variable in $\{+1, -1\}$ with probability $\{\frac{1}{2}, \frac{1}{2}\}$. For keeping the row sum and the column sum of $\mathbf{B}$ constant, the algorithm randomly selects another three elements $b_{j,l'}, b_{j',l}$ and $b_{j',l'}$ $(j' \neq j, l' \neq l)$ from $\mathbf{B}$ and sets $b_{j,l'} := b_{j,l'} - \xi$, $b_{j',l} := b_{j',l} - \xi$ and $b_{i',j'} := b_{j',l'} + \xi$. Notably, it must be ensured that the resulting matrix elements satisfy $b_{j,l} \geq 0, \forall j, l$.

4.3.4. Evaluation

The number of satisfied disjunctive arcs is adopted as the objective function of SA in this stage. Therefore, the evaluation of a solution is formally described as

$$f(\mathbf{B}) = \sum_{(i,j)\in\hat{D}} z_{(i,j)},$$

where $z_{(i,j)} = 1$ if the derived arc (i, j) is satisfied and $z_{(i,j)} = 0$ otherwise.

For example, the decomposition policy shown in Figure 2 has an objective value of

$$f(\mathbf{B}_0) = z_{(1,8)} + z_{(4,2)} + z_{(3,5)} + z_{(6,8)} + z_{(3,9)} = 4,$$

because the directed disjunctive arcs are all satisfied except $(6, 8)$.

4.4. Solving each subproblem

In this paper, the process of solving each subproblem is equivalent to optimizing the sequence of the operations involved in this subproblem (O_l). To this end, a simulated annealing algorithm is designed. The main steps to solve the l-th subproblem are described as follows.

4.4.1. Encoding

The encoding scheme is based on operation priority lists. A solution relates each machine with a priority list of the operations (from O_l) to be processed on this machine. Note that after the decoding process, the actual processing order of the operations in the final feasible schedule may differ from the priority lists.

4.4.2. Initialization

The initial solution is provided by the EDD dispatching rule, where the due date of each operation is taken as h_i.

4.4.3. New solution generation

In order to generate a new solution from the current solution, we must first identify a machine and then somehow alter the operation list related with the selected machine. Hence, machine selection is a key step in this process. We hope to select the machines which are scheduled not very well and therefore have relatively larger room for further improvement.

To achieve this aim, we define an index Ψ_k for each machine k to evaluate the scheduling efficiency of this machine:

$$\Psi_k = \sum_{i\in O_{kl}} (c_i - h_i)^+,$$

where O_{kl} is the set of operations to be processed by machine k in O_l; c_i is the completion time of operation i in the current schedule; $x^+ = \max\{x, 0\}$. Ψ_k reflects the scheduling performance of machine k, and a larger value of Ψ_k suggests machine k is currently scheduled less satisfactorily. Therefore, if we focus more of the subsequent optimization efforts on those machines with higher Ψ values, it is more likely to obtain improvement. So the proportional method (a.k.a. roulette wheel selection) is adopted here for the selection of a machine, i.e., machine k is selected with probability $Prob_k = \Psi_k / \sum_{k=1}^{m} \Psi_k$.

Finally, the "SWAP" operator is used to exchange the positions of two randomly chosen operations in the selected machine's priority list, and then a new solution is produced.

4.4.4. Evaluation

The decoding process iteratively schedules the ready operations in the current subproblem according to their priority order indicated by the solution and as early as possible, on the basis of the partial schedule obtained in the previous subproblems $O_1, \ldots, O_{l-1}$ (a data structure similar to the Gantt chart is used to record the partial schedules obtained in previous subproblems).

Taking the instance shown in Figure 2 as an example, we can see the incremental process of solving each subproblem. After subproblem 1 is solved, the positions of operations 1 and 4 (light grey) should be fixed, which form the basis for solving subproblem 2. After subproblem 2 is solved, the positions of operations 8, 2, 7, 3 and 5 (medium grey) should be fixed, which, together with the previously fixed operations 1 and 4, form the basis for solving subproblem 3. Once subproblem 3 has been solved, the positions of operations 6 and 9 (dark grey) should be fixed. Now, we could obtain a complete solution to the original problem, as shown in Figure 3.

The evaluation of a solution is based on the local objective function defined as

$$L_{\max}^{\text{local}} = \max_{j=1}^{n} \{c_{\omega_l(j)} - h_{\omega_l(j)}\},$$

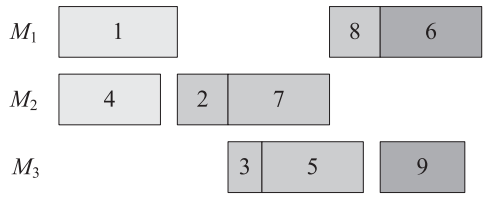

Figure 3. Solving the three subproblems successively

where $\omega_l(j)$ denotes the last operation in the technological route of job j that belongs to the current subproblem O_l. Taking the case described in Figure 3 as an example, when we are solving subproblem 2, the local objective function should be evaluated based on operations 3, 5 and 8, that is, $\omega_2(1) = 3, \omega_2(2) = 5, \omega_2(3) = 8$.

5. Computational experiments

In this section, some JSSP benchmark instances from the OR-Library [26] are used to evaluate the performance of DOASA. Since this research is aimed at large-scale problems, only the instances that include no less than 300 operations are tested. These 27 instances belong to the LA, ABZ, SWV and YN classes. Moreover, in order to adapt the instances for the $L_{\max}$ objective function considered here, the due date of each job is set based on its total processing time as $d_j = \lfloor f \times \sum_{i \in J_j} p_i \rfloor$, where $\sum_{i \in J_j} p_i$ is the sum of processing times of all the operations that constitute job j, and $f \in \{1.3, 1.4, 1.5\}$ is the due date tightness factor.

In the computational experiments, the parameters for SA are set as: the initial acceptance probability $p_r = 0.8$; the temperature reduction ratio $\lambda = 0.95$; the number of inner iterations (number of samples obtained at each temperature) $I_i = 20$; the number of outer iterations (number of temperature stages) $I_o = 50$. The parameters for the CP-based preprocessor is set as: the local search iteration number $w = 100$; the step size for locating upper bound $\Delta = 3$. Meanwhile, the number of subproblems for each instance is fixed according to some preliminary computational results: for the instances with 500 operations (SWV11–SWV20), $p = 6$ is adopted; for the remaining instances with 300 or 400 operations, $p = 4$ is adopted. Once p is fixed, the number of operations in each subproblem is

$$\alpha_l = \begin{cases} \lfloor \frac{|O|}{p+1} \rfloor, & l = 1, \ldots, p-1 \\ |O| - (p-1) \times \lfloor \frac{|O|}{p+1} \rfloor, & l = p \end{cases}$$

The last subproblem contains more operations in order to facilitate the final adjustment of the schedule for improving the objective value.

In order to examine the effectiveness of the presented approach, DOASA is compared with the following methods:

- The genetic algorithm (GA) applying operation-based encoding scheme and heuristic initialization;
- The fast TS/SA hybrid algorithm which combines the advantages of tabu search and simulated annealing [27];

In the GA, the initial population is generated by applying various effective dispatching rules, including SPT, EDD, MDD, etc. [28]. The parameters of GA are set as: the population size is 100, the maximum generation number is 500 (or determined by the exogenous limit on computational time), the crossover probability is 0.8 and the mutation probability is 0.1. The parameters of TS/SA are determined according to the original literature [27].

All the algorithms are implemented in Visual C++ 7 and tested on an Intel Core i5-750 / 3GB RAM / Windows 7 platform. To make the comparison meaningful, we keep the running time of each algorithm identical. In each trial, we run DOASA first and record its computational time as CT, and then run GA and TS/SA under the time limit of CT (which controls the realized number of iterations for each algorithm).

Table 1 presents the results for wide due date settings ($f = 1.5$), Table 2 presents the results for moderate due date settings ($f = 1.4$), and Table 3 presents the results for tight due date settings ($f = 1.3$). In the tables, the first two columns list the instance names and their sizes in the format of $n \times m$. For each instance, all the algorithms are allowed to run 10 independent times. Since absolute objective values are unimportant for comparison purposes, the focus here is put on relative values. π_b is used to characterize the improvement (in percentage) of the best solution obtained by DOASA during the 10 runs over the best solution by the comparative method. Formally, we define

$$\pi_b(a, \mathcal{I}) = \frac{L_{\max}^{\text{best}}(a, \mathcal{I}) - L_{\max}^{\text{best}}(\text{DOASA}, \mathcal{I})}{L_{\max}^{\text{best}}(\text{DOASA}, \mathcal{I})},$$

where $L_{\max}^{\text{best}}(a, \mathcal{I})$ is the best objective value for instance $\mathcal{I}$ obtained by the algorithm a ($a \in \{\text{GA, TS/SA, DOASA}\}$) in the 10 runs. Defined similarly, π_m denotes the percentage improvement of the mean objective value obtained by DOASA during the 10 runs over the mean objective value by the comparative method (detailed formula omitted here). The maximum lateness $L_{\max}$ remains positive even under $f = 1.5$ so that the relative values (π_b, π_m) are meaningful and comparable.

The results in Tables 1-3 reveal the effectiveness of DOASA. The following remarks can be made regarding the algorithm's performance.

(1) At each due date level, the superiority of DOASA over the comparative methods is more significant for the larger instances (SWV11–SWV20 with $p = 6$) than for the smaller instances (with $p = 4$). This can be interpreted as evidence that decomposing a large JSSP instance into smaller subproblems will make the search process more efficient during optimization.

(2) It is observed that for most instances, $\pi_m > \pi_b$ holds, which means the improvement on mean objective value is usually greater than the improvement on best objective value. So it is reasonable to conclude that the stability of DOASA in different executions is considerably stronger than the comparative algorithms. The robustness is an advantage brought by the decomposition-based optimization framework which builds the final schedule in an incremental way.

Instance	Size	GA		TS/SA	
		π_b	π_m	π_b	π_m
LA31	30×10	2.13	1.99	0.65	1.91
LA32	30×10	1.80	1.92	1.40	2.73
LA33	30×10	2.27	2.16	1.84	2.16
LA34	30×10	2.12	2.33	1.99	2.71
LA35	30×10	2.17	2.44	0.22	0.56
ABZ7	20×15	2.31	2.93	1.36	1.82
ABZ8	20×15	2.35	2.96	1.94	3.96
ABZ9	20×15	2.06	2.41	1.12	3.13
SWV06	20×15	2.46	2.73	1.14	3.50
SWV07	20×15	2.87	3.66	2.08	2.06
SWV08	20×15	3.22	3.06	1.74	2.34
SWV09	20×15	3.58	3.76	−0.51	1.73
SWV10	20×15	3.07	4.04	1.43	2.54
YN1	20×20	3.76	3.63	2.08	3.16
YN2	20×20	3.43	4.18	1.74	2.66
YN3	20×20	4.15	3.73	2.66	4.01
YN4	20×20	4.23	3.79	2.53	3.29
SWV11	50×10	4.14	4.67	3.97	6.62
SWV12	50×10	3.91	5.14	3.19	5.84
SWV13	50×10	4.46	4.93	3.62	9.38
SWV14	50×10	4.98	5.26	3.88	6.21
SWV15	50×10	5.29	5.29	4.35	7.92
SWV16	50×10	3.80	5.06	4.48	8.24
SWV17	50×10	5.20	5.82	2.59	5.67
SWV18	50×10	5.13	5.65	3.17	6.46
SWV19	50×10	5.57	6.68	2.42	6.34
SWV20	50×10	5.45	6.29	3.55	5.76
avg.		3.55	3.94	2.25	4.17

Table 1. Relative improvement (%) of DOASA over comparative methods under $f = 1.5$

Instance	Size	GA		TS/SA	
		π_b	π_m	π_b	π_m
LA31	30×10	2.93	2.43	0.78	2.64
LA32	30×10	2.71	2.07	1.52	3.50
LA33	30×10	2.61	2.42	2.24	2.18
LA34	30×10	2.77	2.81	2.42	3.42
LA35	30×10	2.47	3.19	−0.18	0.84
ABZ7	20×15	3.50	3.89	1.40	2.06
ABZ8	20×15	3.48	2.85	2.13	4.76
ABZ9	20×15	3.13	2.63	1.26	3.60
SWV06	20×15	3.61	3.41	1.34	4.28
SWV07	20×15	4.05	4.31	2.69	2.92
SWV08	20×15	4.00	3.54	2.01	2.85
SWV09	20×15	4.72	3.57	0.57	2.03
SWV10	20×15	3.69	4.57	1.91	3.09
YN1	20×20	4.97	5.26	2.65	3.68
YN2	20×20	3.77	5.09	2.37	3.23
YN3	20×20	5.58	5.04	3.05	5.05
YN4	20×20	5.42	5.26	2.49	4.63
SWV11	50×10	6.76	5.72	4.64	9.40
SWV12	50×10	5.07	6.44	3.88	6.95
SWV13	50×10	7.64	5.52	3.84	11.31
SWV14	50×10	5.31	6.67	4.46	7.68
SWV15	50×10	7.47	6.40	5.03	9.05
SWV16	50×10	5.34	5.48	5.29	9.55
SWV17	50×10	5.99	5.79	2.93	6.75
SWV18	50×10	7.33	6.62	3.52	7.49
SWV19	50×10	8.38	6.76	2.61	8.85
SWV20	50×10	7.04	8.41	3.72	6.15
avg.		4.81	4.67	2.61	5.11

Table 2. Relative improvement (%) of DOASA over comparative methods under $f = 1.4$

(3) By comparing the average π_b of GA and TS/SA, we may find that TS/SA performs better than GA in terms of the best solution quality. This can be explained by the fact that the combination of TS and SA significantly enhances the exploitation ability of the search algorithm. Then, if we compare the average π_m of GA and TS/SA, we find that GA outperforms TS/SA in terms of the mean solution quality. Therefore, in spite of its weakness in intensified search, GA performs more stably in different runs due to its population-based searching behaviour.

(4) By comparing the three tables, it is apparent that the relative improvement of DOASA over the comparative algorithms is larger in the case of a tighter due date (i.e., smaller f). Such a difference in due date robustness also reflects the effectiveness of the decomposition procedure based on constraint propagation. When the due dates are tighter (i.e., $UB + d_i - p_i$ is smaller), more disjunctive arc directions can be derived by constraint propagation, which helps to generate a more useful and reliable decomposition policy.

The computational time consumed by DOASA when solving the four representative instances is shown in Figure 4. The due date tightness affects the process of constraint propagation and operation decomposition, so the time needed for handling tight due dates is higher than for wide due dates.

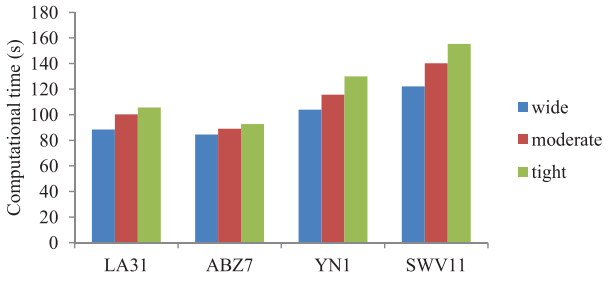

Figure 4. The computational time consumed by DOASA

To test the impact of subproblem number (p) on the final solution quality, the results obtained by DOASA under different p values for two selected instances (SWV06 and SWV11) are recorded and shown as relative percentage values in Figure 5. In this figure, we define

$$R_p = \frac{L_{\max}^{\text{mean}}(p) - L_{\max}^{\text{mean}}(p^*)}{L_{\max}^{\text{mean}}(p^*)},$$

where $L_{\max}^{\text{mean}}(p)$ denotes the obtained mean objective value in 10 runs when the subproblem number is set as p, and $p^* = \arg\min_{p \in \mathcal{P}} L_{\max}^{\text{mean}}(p)$ is the optimal subproblem number within the considered range ($\mathcal{P}$) of possible p values.

Figure 5 suggests the subproblem number in DOASA can affect the solution quality to a noticeable extent. We have $p^* = 4$ for the instance SWV06 and $p^* = 6$ for the instance SWV11. When the actual subproblem number is far less than p^*, the optimization result is much poorer because

Instance	Size	GA		TS/SA	
		π_b	π_m	π_b	π_m
LA31	30 × 10	2.39	2.90	2.72	5.59
LA32	30 × 10	2.47	2.87	2.94	4.12
LA33	30 × 10	2.34	3.21	2.14	5.06
LA34	30 × 10	2.90	3.51	4.08	5.38
LA35	30 × 10	2.92	3.49	3.00	7.54
ABZ7	20 × 15	3.16	3.35	4.40	6.53
ABZ8	20 × 15	3.50	4.11	5.68	7.88
ABZ9	20 × 15	3.21	4.24	1.69	5.72
SWV06	20 × 15	3.02	3.51	3.08	3.56
SWV07	20 × 15	3.67	4.13	2.32	4.19
SWV08	20 × 15	3.54	4.03	3.64	4.46
SWV09	20 × 15	4.17	4.52	2.48	4.16
SWV10	20 × 15	3.00	4.05	3.98	4.11
YN1	20 × 20	3.85	5.40	4.84	7.35
YN2	20 × 20	3.86	4.79	4.26	6.68
YN3	20 × 20	3.29	5.10	1.89	5.19
YN4	20 × 20	4.44	5.07	3.09	7.21
SWV11	50 × 10	5.67	5.70	5.40	11.90
SWV12	50 × 10	4.60	5.53	3.07	10.41
SWV13	50 × 10	5.68	6.90	5.06	8.23
SWV14	50 × 10	6.40	6.54	5.56	12.05
SWV15	50 × 10	5.57	7.58	6.65	11.29
SWV16	50 × 10	5.39	6.18	4.24	9.28
SWV17	50 × 10	5.84	6.96	5.84	9.14
SWV18	50 × 10	5.26	7.10	3.21	8.41
SWV19	50 × 10	6.23	7.76	7.52	9.69
SWV20	50 × 10	8.21	8.19	4.85	8.75
avg.		4.24	5.06	3.99	7.18

Table 3. Relative improvement (%) of DOASA over comparative methods under $f = 1.3$

the benefit of decomposition has not been fully utilized. On the other extreme, if the adopted subproblem number is unduly large, the solution quality also deteriorates because the additional preferential order imposed by the decomposition procedure can reduce the potential search space, thus limiting the subsequent optimization process for each subproblem.

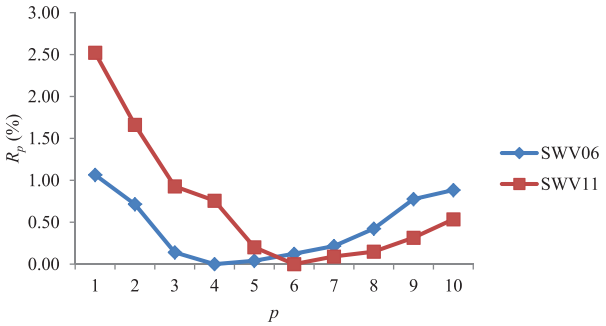

Figure 5. The impact of p on solution quality

6. Conclusion

A decomposition-based optimization algorithm is proposed in this paper for large-sized job shop scheduling problems with the maximum lateness criterion. The theory of constraint propagation is utilized as a preprocessor for the pending problem instance so that a number of disjunctive arcs can be correctly oriented. Simulated annealing is adopted as the main framework in the two optimization procedures, i.e., the search for a promising decomposition policy and the solving of each subproblem. Experiments have been carried out for adapted JSSP benchmark instances. The performance of the proposed algorithm surpasses the competing methods (under the same computational time limit), especially for larger instances and under tighter due date settings. The results verify the effectiveness and efficiency of both the decomposition procedure and the subproblem solution procedure. In the future research, parallel computing (e.g., based on GPU) techniques may be considered as a method for accelerating the execution of the proposed algorithm. In this way, the subproblems can be solved in parallel once they have been determined by the decomposition procedure, which will shorten the running time considerably.

7. Acknowledgements

The paper is supported by the National Natural Science Foundation of China (61104176), the Science and Technology Project of Jiangxi Provincial Education Department (GJJ12131), the Social Sciences Research Project of Jiangxi Provincial Education Department (GL1236), and the Educational Science Research Project of Jiangxi Province (12YB114).

8. References

[1] V. Vinod and R. Sridharan. Scheduling a dynamic job shop production system with sequence-dependent setups: An experimental study. *Robotics and Computer-Integrated Manufacturing*, 24(3):435–449, 2008.

[2] T. Çakar, R. Köker, and Y. Sarı. Parallel robot scheduling to minimize mean tardiness with unequal release date and precedence constraints using a hybrid intelligent system. *International Journal of Advanced Robotic Systems*, 9, 2012.

[3] R. Ramasesh. Dynamic job shop scheduling: a survey of simulation research. *Omega – The International Journal of Management Science*, 18(1):43–57, 1990.

[4] I. Ahmed and W. W. Fisher. Due date assignment, job order release, and sequencing interaction in job shop scheduling. *Decision Sciences*, 23(3):633–647, 1992.

[5] I. Sabuncuoglu and A. Comlekci. Operation-based flowtime estimation in a dynamic job shop. *Omega – The International Journal of Management Science*, 30(6):423–442, 2002.

[6] S. Q. Liu and E. Kozan. Scheduling trains with priorities: a no-wait blocking parallel-machine job-shop scheduling model. *Transportation Science*, 45(2):175–198, 2011.

[7] J. Carlier and E. Pinson. An algorithm for solving the job shop problem. *Management Science*, 35:164–176, 1989.

[8] D. Sculli. Priority dispatching rules in job shops with assembly operations and random delays. *Omega – The International Journal of Management Science*, 8(2):227–234, 1980.

[9] J. J. Kanet and J. C. Hayya. Priority dispatching with operation due dates in a job shop. *Journal of Operations Management*, 2(3):167–175, 1982.
[10] K. R. Baker. Sequencing rules and due-date assignments in a job shop. *Management Science*, 30(9):1093–1104, 1984.
[11] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391–401, 1988.
[12] E. Balas and A. Vazacopoulos. Guided local search with shifting bottleneck for job shop scheduling. *Management Science*, 44(2):262–275, 1998.
[13] S. Q. Liu and E. Kozan. A hybrid shifting bottleneck procedure algorithm for the parallel-machine job-shop scheduling problem. *Journal of the Operational Research Society*, 63(2):168–182, 2012.
[14] 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 & Operations Research*, 38(5):854–867, 2011.
[15] G. I. Zobolas, C. D. Tarantilis, and G. Ioannou. A hybrid evolutionary algorithm for the job shop scheduling problem. *Journal of the Operational Research Society*, 60(2):221–235, 2009.
[16] V. Sels, K. Craeymeersch, and M. Vanhoucke. A hybrid single and dual population search procedure for the job shop scheduling problem. *European Journal of Operational Research*, 215(3):512–523, 2011.
[17] Z. C. Zhu, K. M. Ng, and H. L. Ong. A modified tabu search algorithm for cost-based job shop problem. *Journal of the Operational Research Society*, 61(4):611–619, 2010.
[18] R. Zhang, S. Song, and C. Wu. A two-stage hybrid particle swarm optimization algorithm for the stochastic job shop scheduling problem. *Knowledge-Based Systems*, 27:393–406, 2012.
[19] J. B. Sidney. Decomposition algorithms for single machine sequencing with precedence relations and deferral costs. *Operations Research*, 23(2):283–298, 1975.
[20] M. H. Bassett, J. F. Pekny, and G. V. Reklaitis. Decomposition techniques for the solution of large-scale scheduling problems. *AIChE Journal*, 42(12):3373–3387, 1996.
[21] D. Sun and R. Batta. Scheduling larger job shops: a decomposition approach. *International Journal of Production Research*, 34(7):2019–2033, 1996.
[22] M. Singer. Decomposition methods for large job shops. *Computers & Operations Research*, 28(3):193–207, 2001.
[23] S. D. Wu, E. S. Byeon, and R. H. Storer. A graph-theoretic decomposition of the job shop scheduling problem to achieve scheduling robustness. *Operations Research*, 47(1):113–124, 1999.
[24] U. Dorndorf, E. Pesch, and T. Phan-Huy. Constraint propagation techniques for the disjunctive scheduling problem. *Artificial Intelligence*, 122(1):189–240, 2000.
[25] T. Phan-Huy. *Constraint propagation in flexible manufacturing*. Springer, 2000.
[26] J. E. Beasley. OR-library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.
[27] C. Y. Zhang, P. G. Li, Y. Q. Rao, and Z. L. Guan. A very fast TS/SA algorithm for the job shop scheduling problem. *Computers & Operations Research*, 35(1):282–294, 2008.
[28] S. Panwalkar and W. Iskander. A survey of scheduling rules. *Operations Research*, 25(1):45–61, 1977.