

Infeasibility test algorithm and fast repair algorithm of job shop scheduling problem

Sun Lu^{1,2} Huang Zhi^{2,3} Zhang Huiming⁴ Gu Wenjun¹

(¹School of Transportation, Southeast University, Nanjing 210096, China)

(²Department of Civil Engineering, Catholic University of America, Washington DC 20064, USA)

(³School of Computer Science, Huazhong University of Science and Technology, Wuhan 430074, China)

(⁴Jinzhong Bureau of Highway Administration, Jinzhong 030600, China)

Abstract: To diagnose the feasibility of the solution of a job-shop scheduling problem (JSSP), a test algorithm based on diagraph and heuristic search is developed and verified through a case study. Meanwhile, a new repair algorithm for modifying an infeasible solution of the JSSP to become a feasible solution is proposed for the general JSSP. The computational complexity of the test algorithm and the repair algorithm is both $O(n)$ under the worst-case scenario, and $O(2|J| + |M|)$ for the repair algorithm under the best-case scenario. The repair algorithm is not limited to specific optimization methods, such as local tabu search, genetic algorithms and shifting bottleneck procedures for job shop scheduling, but applicable to generic infeasible solutions for the JSSP to achieve feasibility.

Key words: infeasibility; job shop scheduling; repairing algorithm

doi: 10.3969/j.issn.1003-7985.2011.01.018

NP-hard (non-deterministic polynomial-time hard), in computational complexity theory, is a class of problems that are, informally, “at least as hard as the hardest problems in NP”. A problem H is NP-hard if and only if there is an NP-complete problem L that is polynomial time Turing reducible to H (i. e., $L \leq_p H$). In other words, L can be solved in polynomial time by an oracle machine with an oracle for H . Informally, we can think of an algorithm that can call such an oracle machine as a subroutine for solving H , and solves L in polynomial time, if the subroutine call takes only one step to compute.

The job shop scheduling problem (JSSP) is an NP-hard problem in the category of combinatorial optimization^[1]. Although many studies have been carried out to deal with the JSSP, the feasibility test of constraints of the JSSP is still an undeveloped research area. The feasibility test of the JSSP answers the concern of the existence of a feasible solution of a JSSP. Whether a set of constraints of the JSSP is feasible or not remains a very critical problem, sometimes even more fundamental than solving the optimization itself. If a JSSP is infeasible, how to repair the constraints so that a feasible so-

lution can be constructed is also of great interest as it deals with many practical issues in the application of the JSSP.

In recent years, a number of methods have been developed to solve the JSSP and to study feasibility of a set of constraints of the JSSP^[2-6]. Most of the existing methods avoid infeasibility when encountered. Few papers have discussed the infeasibility repair with limitations. Murovec and Suhel^[7] proposed an infeasibility repair algorithm during local search in solving a JSSP. The algorithm is limited to repairing only the infeasibility caused by a move of particular kind during local searches, and it is not suitable for repairing the infeasibility of a generic sort. Mattfeld^[8] developed an infeasibility repair algorithm, which is valid only when the optimization method for solving a JSSP is based on genetic algorithms.

Given the limitations of existing algorithms for testing and repairing the infeasibility of a JSSP^[8], we present a feasibility test algorithm and a generic repair algorithm, which are new compared to previous literature, for any infeasible solution of the JSSP. Both algorithms are featured with low complexity and high computational efficiency.

1 Job Shop Scheduling Problem

The JSSP aims at minimizing the processing and completion time of all the jobs subject to the constraints: 1) The sequence of machines for each job being prescribed, and 2) each machine being allowed to process only one job at a time. Let $N = \{0, 1, \dots, n\}$ denote the set of operations (with 0 and n being dummy operations “start” and “finish”), M the set of machines, A the set of pairs of operations constrained by the precedence relations given in 1), and E_k the set of pairs of operations to be performed on machine k , and, therefore, cannot overlap in time, as specified in 2). Further, let d_i denote the (fixed) duration (processing time) and t_i the (variable) start time of operation i . The problem can then be stated as

$$\begin{aligned} & \min t_n \\ \text{s. t. } & t_j - t_i \geq d_i & (i, j) \in A \\ & t_i \geq 0 & i \in N \\ & t_j - t_i \geq d_i \vee t_i - t_j \geq d_j & (i, j) \in E_k; k \in M \end{aligned}$$

A feasible solution to the above-mentioned problem is called a schedule^[9-10]. It is helpful to represent this problem on a disjunctive graph $G = (N, A, E)$, with node set N , conjunctive arc set A , and disjunctive arc set E . Conjunctive arcs represent precedence relations, and pairs of disjunctive arcs correspond to operations that must be processed on the same machine. Fig. 1 illustrates the graph for a JSSP

Received 2010-09-13.

Biography: Sun Lu (1972—), male, doctor, professor, sunl@cua.edu.

Foundation items: The US National Science Foundation (No. CMMI-0408390, CMMI-0644552), the Research Fellowship for International Young Scientists (No. 51050110143), the Fok Ying-Tong Education Foundation (No. 114024), the Natural Science Foundation of Jiangsu Province (No. BK2009015), the Postdoctoral Science Foundation of Jiangsu Province (No. 0901005C).

Citation: Sun Lu, Huang Zhi, Zhang Huimin, et al. Infeasibility test algorithm and fast repair algorithm of job shop scheduling problem[J]. Journal of Southeast University (English Edition), 2011, 27(1): 88 – 91. [doi: 10.3969/j.issn.1003-7985.2011.01.018]

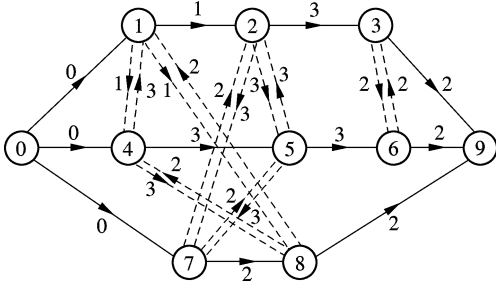


Fig. 1 Disjunctive graph of a job shop scheduling problem

with eight operations (on three jobs) and three machines.

A selection S_k in E_k contains exactly one member of each disjunctive arc pair of E_k . Determining an acyclic section on S_k is equivalent to making a sequence of operations on machines k . A complete selection S consists of the union of selection S_k , one in each E_k , $k \in M$. In the language of disjunctive graphs, the problem is to find an acyclic complete selection $S \subset E$ that minimizes the length of the longest path in the directed graph D_S . The problem can also be described so as to find $\pi = (\pi_1, \pi_2, \dots, \pi_{|M|})$, the processing order of operations on machines, making the diagram $G(\pi) = (N, A \cup E(\pi))$ acyclic and minimizing the length of the longest path in the diagram.

2 Feasibility Test Algorithm

Fig. 2 illustrates the two kinds of precedence relationships for a solution of the JSSP. One is job precedence shown in Fig. 2(a), and the other is machine precedence shown in Fig. 2(b). Any solution (processing order of operations on machines) that satisfies both precedence relationships is a feasible solution. Any solution that does not satisfy both precedence relationships is an infeasible one. Fig. 2(c) shows the diagram of the feasible solution π . Here $\pi = (\pi_1, \pi_2, \pi_3) = ((4, 1, 8), (2, 7, 5), (3, 6))$. No cycle exists in the diagram shown in Fig. 2(c).

Fig. 3(a) shows the precedence relationships of the infeasible solution $\pi' = (\pi'_1, \pi'_2, \pi'_3) = ((4, 8, 1), (2, 7, 5), (3, 6))$.

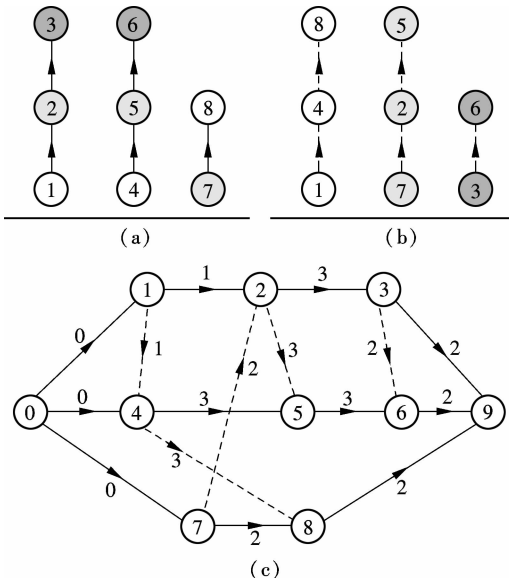


Fig. 2 Feasible solution $\pi = ((1, 4, 8), (7, 2, 5), (3, 6))$.

(a) Job precedence relationships for π ; (b) Machine precedence relationships for π ; (c) Diagram $G(\pi) = (N, A \cup E(\pi))$ for π

$(3, 6)$). The diagram of the infeasible solution π' is shown in Fig. 3(b), where the cycle $1 \rightarrow 2 \dots \rightarrow 7 \rightarrow 8 \dots \rightarrow 1$ exists. The precedence cycle $1 \rightarrow 2 \dots \rightarrow 7 \rightarrow 8 \dots \rightarrow 1$ can be seen in Figs. 3(a) and (b). Given an instance $G(\pi) = (N, A \cup E(\pi))$ and an operation v , let $jp(v)$ and $js(v)$ denote the immediate predecessor and successor of v in A , if they exist. $mp(v)$ and $ms(v)$ denote the immediate predecessor and successor of v in π , if they exist. Denote J to be the set of jobs. Denote N_i to be the sequence of operations belonging to job J_i . Denote $N_i(j)$ to be the j -th operation in $|N_i|$, and $|N_i|$ the amount of operations belonging to job J_i . Let $|\pi_i|$ be the amount of operations processed on machine i . For the example shown in Fig. 3, $N_1 = (1, 2, 3)$, $N_2 = (4, 5, 6)$, $N_3 = (7, 8)$. $N_1(1) = 1$, $N_1(2) = 2$, $N_1(3) = 3$, $N_2(1) = 4$, $N_3(1) = 7$.

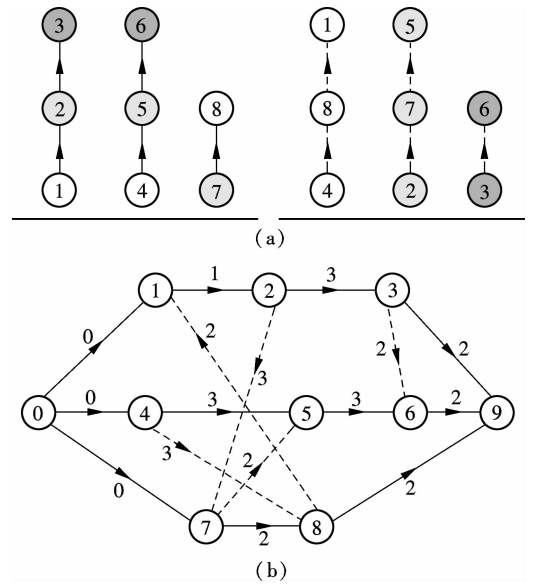


Fig. 3 Infeasible solution $\pi' = ((4, 8, 1), (2, 7, 5), (3, 6))$. (a) The precedence relationships for solution π' ; (b) Diagram $G(\pi') = (N, A \cup E(\pi'))$

For a given processing order of operations on machines, we propose the following algorithm to test whether a solution is a feasible solution of the JSSP or not.

2.1 The infeasibility test algorithm T

N' is the set of operations that have not been labeled. O_j , O_M and K are subsets of N' ; O_j is the set of operations which have no job predecessors in N' ; O_M is the set of operations which have no job predecessors in π' ; and K is the set of operations which have no job predecessors in N' .

Step 1 $N' = N - \{0\} - \{n\}$;

$O_j = \{N_1(1), N_2(1), N_3(1), \dots, N_{|J|}(1)\}$;

$O_M = \{\pi_1(1), \pi_2(1), \pi_3(1), \dots, \pi_{|M|}(1)\}$.

Step 2 If $O_j \cap O_M \neq \emptyset$, then do the followings:

$K = O_j \cap O_M$; label each operation in K .

$O_j = O_j - K$; $O_M = O_M - K$; $N' = N' - K$.

For each $i \in K$: if $js(i)$ exists, $O_j = O_j \cup \{js(i)\}$; if $ms(i)$ exists, $O_M = O_M \cup \{ms(i)\}$.

Go to step 2.

Step 3 If $O_j \cup O_M = \emptyset$, FEASIBLE: = YES; otherwise FEASIBLE: = NO.

Stop.

Theorem 1 For any solution π , the diagram $G(\pi)$ is acyclic if and only if $O_J \cup O_M = \emptyset$ is satisfied in step 3 of algorithm T.

Proof For the case of $O_J \cup O_M = \emptyset$ in step 3 of algorithm T, we denote the initial value of N' , O_J , and O_M with N'_0 , $O_{J(0)}$, and $O_{M(0)}$; the sequence of values of K , N' , O_J and O_M obtained in step 2 with $(K_1, K_2, \dots, K_{\text{end}-1}, K_{\text{end}})$, $(O_J(1), O_J(2), \dots, O_J(\text{end}))$, $(O_M(1), O_M(2), \dots, O_M(\text{end}))$ and $(N'_1, N'_2, \dots, N'_{\text{end}})$. Then $O_{J(\text{end})}$, $O_{M(\text{end})}$, $N'_{\text{end}} = \emptyset$. Obviously $K_{\text{end}} = \emptyset$, otherwise step 3 will not be executed according to algorithm T. According to algorithm T, $K_i \cap K_j = \emptyset (0 < i < j < \text{end})$, and $\{0\} \cup K_1 \cup K_2 \cup \dots \cup K_{\text{end}-1} \cup \{n\} = N$. $O_{J(i-1)} (1 \leq i \leq \text{end})$ is the set of operations which have no job predecessors in N'_{i-1} , and $O_{M(i-1)}$ is the set of operations which have no job predecessors in π'_{i-1} . Since $K_i = O_{J(i-1)} \cap O_{M(i-1)}$, any operation in K_i has no job predecessor in $N'_{i-1} = K_i \cup K_{i+1} \cup \dots \cup K_{\text{end}-1}$. Therefore, the diagram $G(\pi)$ can be redrawn with operations in K_i being placed to the left of operations in $K_{i+1} (1 \leq i \leq \text{end} - 2)$, operation 0 being placed to the left of all other operations, and operation n being placed to the right of all other operations. After these operations, the direction of each arc in the diagram $G(\pi)$ is from left to right. Thus $G(\pi)$ is acyclic.

For the case of $O_J \cup O_M \neq \emptyset$ in step 3 of algorithm T, obviously, $O_J \cap O_M = \emptyset$; otherwise, algorithm T will not execute step 3. $N' \neq \emptyset$, because if $N' = \emptyset$, then $O_J = \emptyset$, $O_M = \emptyset$ and $O_J \cup O_M = \emptyset$ which is a contradiction. O_J is the set of operations which have no job predecessors in N' , so $O_J \neq \emptyset$. Similarly $O_M \neq \emptyset$. For an operation b_1 in O_J , b_1 has a machine successor b_2 in O_M . Because if b_1 has no machine successor b_2 in O_M , it leads to $b_1 \in O_M$ and $b_1 \in O_M \cap O_J$ which is a contradiction to $O_J \cap O_M = \emptyset$. Similarly, b_2 has a job successor b_3 in O_J . Therefore, in diagram $G(\pi)$, there exists a path from b_3 to b_2 to b_1 , denoted with (b_3, b_2, b_1) . In this way, we can prove that the path $(b |N'| + 1, b |N'|, \dots, b_3, b_2, b_1)$ exists in $G(\pi)$. Since $b_i \in N' (1 \leq i \leq |N'| + 1)$, there exists $b_j = b_k (1 \leq j < k \leq |N'| + 1)$. Therefore, the cycle path $(b_k, b_{k-1}, \dots, b_{j+1}, b_j)$ exists in $G(\pi)$. Thus $G(\pi)$ is cyclic. This completes the proof.

For the given solution $\pi = ((1, 4, 8), (7, 2, 5), (3, 6))$ shown in Fig. 2, $N'_0 = \{1, 2, 3, \dots, 8\}$, $O_{J(0)} = \{1, 4, 7\}$, $O_{M(0)} = \{1, 7, 3\}$. $K_1, K_2, \dots, K_{\text{end}-1} = \{1, 7\}, \{2, 4\}, \{3, 5, 8\}, \{6\}$ (see Fig. 4).

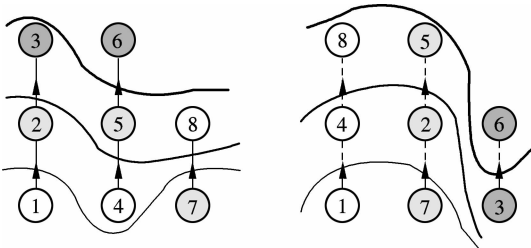


Fig. 4 $K_1, K_2, \dots, K_{\text{end}-1}$ for the solution $\pi = ((1, 4, 8), (7, 2, 5), (3, 6))$

2.2 Complexity of test algorithm T

N' is the variable that we add to make the algorithm easy to understand. It can be ignored in the implementation of the algorithm.

1) The best-case complexity. In the best case, $O_J \cap O_M = \emptyset$ for the first time step 2 is executed; i. e., $\{N_1(1), N_2(1), N_3(1), \dots, N_{|J|}(1)\} \cap \{\pi_1(1), \pi_2(1), \pi_3(1), \dots, \pi_{|M|}(1)\} = \emptyset$. This can be accomplished by $|J|$ times checking whether $N_i(1) (1 \leq i \leq |J|)$ belong to $\{\pi_1(1), \pi_2(1), \pi_3(1), \dots, \pi_{|M|}(1)\}$. Additionally, the initializations for the marking operations in $\{N_1(1), N_2(1), N_3(1), \dots, N_{|J|}(1)\}$ and operations in $\{\pi_1(1), \pi_2(1), \pi_3(1), \dots, \pi_{|M|}(1)\}$ need $(|J| + |M|)$ assignments. Therefore, $O(2|J| + |M|)$ time is taken for the test algorithm in the best case.

2) The worst-case complexity. In the worst case, for computing the value of $K_i (2 \leq i \leq \text{end})$, we can use a clever implementation. Since each member of K_i is either $js(v)$ or $ms(v) (v \in K_{i-1})$, K_i can be achieved by checking each $js(v)$ and $ms(v) (v \in K_{i-1})$. In the checking, if the operation ($js(v)$ or $ms(v)$) is already marked as a member of O_J or O_M , insert the operation into K ; otherwise mark it as a member of O_J or O_M . In the worst case, the algorithm stops when $O_J \cap O_M = \emptyset$ and $O_J = O_M = \emptyset$. Each operation in $N'_0 - \{N_1(1), N_2(1), N_3(1), \dots, N_{|J|}(1)\} \cup \{\pi_1(1), \pi_2(1), \pi_3(1), \dots, \pi_{|M|}(1)\}$ is both $js(v)$ and $jm(w) (v, w \in N'_0)$, so it is checked twice. The checking includes the comparison and assignment. Each operation in $\{N_1(1), N_2(1), N_3(1), \dots, N_{|J|}(1)\} \cup \{\pi_1(1), \pi_2(1), \pi_3(1), \dots, \pi_{|M|}(1)\}$ is checked one or two times. Therefore, the worst-case complexity is $O(n)$.

3 Infeasibility Repair Algorithm

We further design an infeasibility repair algorithm R to repair any infeasible solution π to achieve a feasible solution.

Algorithm R Denote μ_i to be the machine on which operation i is processed. N' is the set of operations that have not been labeled. O_J is the set of operations which have no job predecessors in N' , O_M is the set of operations which have no job predecessors in N' , and K is the set of operations which have no predecessors in N' .

Step 1 $N' := N - \{0\} - \{n\}$;

$O_J := \{N_1(1), N_2(1), N_3(1), \dots, N_{|J|}(1)\}$;

$O_M := \{\pi_1(1), \pi_2(1), \pi_3(1), \dots, \pi_{|M|}(1)\}$.

Step 2 If $O_J \cap O_M \neq \emptyset$, then do the following:

$K := O_J \cap O_M$; label each operation in K ;

$O_J := O_J - K$; $O_M := O_M - K$; $N' := N' - K$;

For each $i \in K$: if $js(i)$ exists, $O_J := O_J \cup \{js(i)\}$; if $ms(i)$ exists, $O_M := O_M \cup \{ms(i)\}$.

Go to step 2.

Step 3 If $O_J \cup O_M \neq \emptyset$, take any operation k from O_J , rearrange its position in π_{μ_k} ; insert it just before $\pi_{\mu_i}(c_{\mu_k})$ (c_{μ_k} is the sequence number of the operation, which is processed on machine μ_k and belongs to O_M , in machine μ_k), go to step 2; otherwise, stop.

In step 3, inserting k just before $\pi_{\mu_i}(c_{\mu_k})$ change π_{μ_i} from

$(\pi_{\mu_i}(1), \pi_{\mu_i}(2), \dots, \pi_{\mu_i}(c_{\mu_k}-1), \pi_{\mu_i}(c_{\mu_k}), \pi_{\mu_i}(c_{\mu_k}+1) \dots, k, \dots)$ to $(\pi_{\mu_i}(1), \pi_{\mu_i}(2), \dots, \pi_{\mu_i}(c_{\mu_k}-1), k, \pi_{\mu_i}(c_{\mu_k}+1), \dots, \pi_{\mu_i}(c_{\mu_k}), \dots)$.

For infeasible solution $\pi' = ((4, 8, 1), (2, 7, 5), (3, 6))$ shown in Fig. 3, utilizing step 3 of algorithm R, it can rearrange any operation of $\{1, 5, 7\}$. Fig. 5 shows the case that operation 5 is rearranged just before operation 2 in π'_2 . Rearranging operation 5 can be regarded as two reversals: $(2, 7)$ and $(7, 5)$, which obviously changes the sequence more than rearranging operation 1 or operation 7. Considering this, a modified repairing algorithm R can be obtained by taking the operation k satisfying $b_k - c_{\mu_k} = \min \{b_i - c_{\mu_i} \mid i \in O_j\}$ instead of any operation from O_M . Here, μ_i is the machine on which operation i is processed, and b_i is of the sequence number operation i in machine μ_i . c_x is the sequence number of the operation, which is processed on machine x and belongs to O_M , in machine x . For the case shown in Fig. 5, $\mu_1 = 1$, $b_1 = 3$, $c_{\mu_1} = 2$, $\mu_5 = 2$, $b_5 = 3$, $c_{\mu_5} = 1$, $\mu_7 = 2$, $b_7 = 2$, $c_{\mu_7} = 1$.

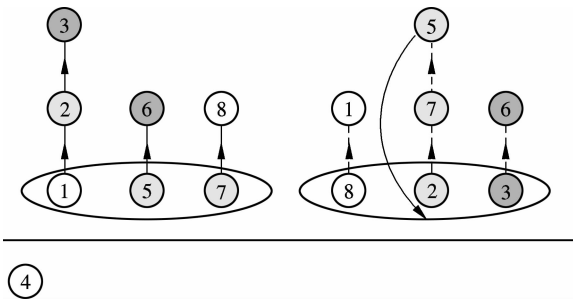


Fig. 5 The rearrangement of operation 5 in repairing procedure

In the worst case of algorithm R, all the operations in $N - \{0, n\}$ need rearranging (described in step 3) except those in $\{\pi_1(1), \pi_2(1), \pi_3(1), \dots, \pi_{|M|}(1)\}$. Each operation of rearrangement can be done in $O(1)$ time. Therefore the worst-case complexity of algorithm R is $O(n)$.

4 Conclusion

This paper addresses the infeasibility issue of the JSSP. A theorem on the infeasibility testing algorithm is proposed

and proved to guarantee the correctness of the algorithm. The complexity of the test algorithm T in the best case is $O(2|J| + |M|)$, and $O(n)$ in the worst case. A repair algorithm is also proposed with the worst-case complexity of $O(n)$. The repair algorithm is not limited to specific optimization methods, such as local tabu searches, genetic algorithms and shifting bottleneck procedures for job shop scheduling, but applicable to generic infeasible solutions for the JSSP to achieve feasibility.

References

- [1] Garey M R, Johnson D S. *Computers and intractability: a guide to the theory of NP-completeness* [M]. San Francisco: Freeman, 1979.
- [2] Jain A S, Meeran S. Deterministic job-shop scheduling: past, present, and future [J]. *European Journal of Operational Research*, 1999, **113**(2): 390–434.
- [3] Jain A S, Meeran S. A state-of-the-art review of job-shop scheduling techniques [R]. Nethergate, Dundee, UK: Department of Applied Physics, Electronics and Mechanical Engineering of University of Dundee, Scotland, 1998.
- [4] Balas E, Vazacopoulos A. Guided local search with shifting bottleneck for job shop scheduling [J]. *Management Science*, 1998, **44**(2): 262–275.
- [5] Luh P B, Zhao X, Wang Y, et al. Lagrangian relaxation neural networks for job shop scheduling [J]. *IEEE Transactions on Robotics and Automation*, 2000, **16**(1): 78–88.
- [6] Geyik F, Cedimoglu I H. The strategies and parameters of tabu search for job-shop scheduling [J]. *Journal of Intelligent Manufacturing*, 2004, **15**(4): 439–448.
- [7] Murovec B, Suhel P. A repairing technique for the local search of the job-shop problem [J]. *European Journal of Operational Research*, 2002, **153**(1): 220–238.
- [8] Mattfeld D C. *Evolutionary search and the job shop: investigations on genetic algorithms for production scheduling* [M]. Heidelberg, Germany: Physica-Verlag, 1996.
- [9] Adams J, Balas E, Zawack D. The shifting bottleneck procedure for job shop scheduling [J]. *Management Science*, 1988, **34**(3): 391–401.
- [10] Balas E. Machine sequencing via disjunctive graphs: an implicit enumeration algorithm [J]. *Operations Research*, 1969, **17**(6): 941–957.

作业车间调度问题解的不可行性检测算法和快速修复算法

孙 璐^{1,2} 黄 志^{2,3} 张惠民⁴ 顾文钧²

(¹ 东南大学交通学院, 南京 210096)

(² Department of Civil Engineering, Catholic University of America, Washington DC 20064, USA)

(³ 华中科技大学计算机学院, 武汉 430074)

(⁴ 山西省公路局晋中分局, 晋中 030600)

摘要: 为了判别作业车间调度问题的解的可行性, 提出了一种基于图论的启发式判别算法, 并通过实例验证了方法的正确性. 提出了普适于作业车间调度问题的快速修补新算法, 可以对于作业车间调度问题的不可行解进行修正使之变成可行解. 判别算法和修补算法在最不利情形下的计算复杂性均为 $O(n)$, 判别算法在最有利情形下的计算复杂性为 $O(2|J| + |M|)$. 所提出的算法具有很大的灵活性, 对于局部蚂蚁算法、遗传算法以及一般的作业车间调度问题均适用.

关键词: 不可行解; 作业车间调度; 修复算法

中图分类号: TP391