Pergamon

0305-0548(94)E0018-3

Computers Ops Res. Vol. 22, No. 2, pp. 159–172, 1995
Copyright © 1995 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0305-0548/95 $7.00 + 0.00

EFFECTIVE JOB SHOP SCHEDULING THROUGH ACTIVE CHAIN MANIPULATION

DAKE SUN,† RAJAN BATTA‡ and LI LIN§
Department of Industrial Engineering, State University of New York at Buffalo, Buffalo, NY 14260, U.S.A.

(Received November 1993; in revised form February 1994)

Scope and Purpose—The problem of scheduling job shops is widely encountered in manufacturing practice. It has attracted researchers' attention since 1950s. After four decades' research, however, generating good job shop schedules in reasonable time remains a challenging problem, because of the problem's inherent computational complexity. To attack this difficult problem, a new method is developed in this paper, which is based on a local search-based iterative improvement approach. Our computation shows that the method is both effective and efficient—it has generated optimal schedules, for some benchmark problems, in much shorter time than other algorithms previously reported in the literature. In addition, the method is also conceptually simple and easy to implement.

Abstract—A practical yet effective heuristic algorithm is developed in this paper for solving the make-span reduction job shop scheduling problem. The algorithm iteratively improves an existing job shop schedule through exploring the schedule's neighborhood, using a simple active chain manipulation scheme. A Tabu search technique is employed, as part of the active chain manipulation procedure developed in this paper, to prevent the trap of local optimality. Test results show that the algorithm is capable of efficiently generating very good schedules.

1. INTRODUCTION

Most job shop scheduling research efforts have been focused on two approaches: (1) developing optimization procedures, and (2) designing new dispatching rules.

The optimization approach (see, for example, Lageweg *et al.* [1], Carlier and Pinson [2] and Applegate and Cook [3]) emphasizes a systematic exploration of all alternative schedules. Since job shop scheduling problems are so called "NP-hard" problems, optimization procedures have to search a very large decision space, even when decision space reduction schemes, such as a branch and bound procedure, are employed. Despite some recent new developments, such as the work of Applegate and Cook [3], solving a realistic sized job shop scheduling problem to optimality in reasonable time remains a challenging problem. Due to the problem's computational intractability, the optimization approach has so far failed to make a significant impact on the scheduling of jobs on actual shop floors.

In contrast, the exploration of alternative schedules is simply not addressed by the dispatching rule approach. One dispatching rule corresponds to only one schedule. Even when dispatching rules are used in a combined way (see, for example, Blackstone *et al.* [4], Wu and Wysk [5]), only very few alternative schedules can be generated. This makes dispatching rules very efficient in generating schedules, but often at the expense of effectiveness.

The scheduling algorithm developed in this paper follows an *Iterative Improvement Approach.*

†Dake Sun obtained his Ph.D. from SUNY at Buffalo in 1994. His research interests include production planning/control and scheduling. This paper represents a portion of his doctoral dissertation, which he recently completed under the joint supervision of Drs Batta and Lin.

‡Rajan Batta is an Associate Professor and Interim Chair in the Department of Industrial Engineering at SUNY at Buffalo. His research interests are in stochastic modeling in transportation and production systems.

§Li Lin obtained his Ph.D. from Arizona State University in 1989. Since then he has been an Assistant Professor at the Department of Industrial Engineering, SUNY at Buffalo. His research interests include modeling and control of flexible manufacturing systems, computer simulation and integration of CAD/CAM. His work has been supported by DuPont, IDEX/Vibratech, Carborandum and the National Science Foundation.

This approach seeks a middle ground between the two above-mentioned computational extremes, to balance scheduling efficiency and effectiveness. The emphasis of the approach is improvement instead of optimality. With this approach, new schedules are generated by iteratively improving an existing schedule; an alternative schedule space is searched but only in a limited way.

The scheduling procedures following this approach have ranged from a simple Monte Carlo procedure [6] to more sophisticated ones, such as the simulated annealing algorithm [7] and the shifting bottleneck procedure [8]. Here, the scheduling algorithm to be discussed is based on an active chain manipulation scheme developed in this paper, which applies the Tabu search procedure of Glover [9, 10].

The job shop scheduling problem discussed in this paper is a make-span reduction problem described as follows: N jobs are to be processed on M machines; each job follows a prescribed routing; the processing times of operations are also prescribed; and all operations are non-preemptive. The objective of the job shop scheduling problem is to find a schedule having minimum make-span that is the earliest time that all jobs can be completed.

The remainder of this paper is organized as follows: Section 2 introduces active schedules and active chains. Section 3 presents the make-span reduction algorithm. Section 4 compares the active chain manipulation scheme and disjunctive graph-based pairwise exchange. Computational results are shown in Section 5, and Section 6 concludes the paper.

2. ACTIVE SCHEDULES AND ACTIVE CHAINS

The job shop scheduling algorithm developed in this paper generates a job shop schedule by searching an *active schedule* space through *active chain* manipulation. Central to the development of the scheduling algorithm is the generation of active schedules and the construction and manipulation of active chains. In this section, active schedules and active chains are defined and studied.

2.1. Active Schedules

Definition

An *active schedule* is a feasible schedule in which no operation can be started earlier than it is scheduled without delaying the starting time of some other operation.

The concept of active schedule was introduced by Giffler and Thompson [11]. In general, if regular performance measures are considered, the set of active schedules is the smallest dominant set in all schedules [11, 12], i.e. for any non-active schedules, there exists an active schedule that is at least as good as that schedule. Therefore, for scheduling problems with regular performance measures, it is sufficient to consider only active schedules.

The algorithm for generating active schedules was proposed by Giffler and Thompson [11]. The active schedule generation algorithm presented in this paper is from Baker [12], which is a variation of Giffler and Thompson's procedure. It generates one active schedule at a time according to a dispatching rule.

For the description of the algorithm, we first introduce some notation. The active schedule generation algorithm schedules one operation at each step. An operation is said to be *schedulable* if all its predecessor operations are already scheduled. At each step of active schedule generation, all operations so far scheduled are contained in a *partial schedule*. Let (i, j) denote the jth operation of job i; O the set of all operations; PS_n the partial schedule that contains n scheduled operations; S_n the set of schedulable operations at step n, corresponding to a given PS_n; $s_{(i,j)}$ the earliest time at which operation $(i, j) \in S_n$ could be started; and $f_{(i,j)}$ the earliest time at which operation $(i, j) \in S_n$ could be finished, $f_{(i,j)} = S_{(i,j)} + p_{(i,j)}$, where $p_{(i,j)}$ is the processing time of operation (i, j). The active schedule generation algorithm, named as ASG Algorithm, is described as follows.

ASG Algorithm

Step 1. Initialize $n = 0$ and $PS_n = \varnothing$; S_n is initialized to contain all operations without predecessors.

Step 2. Determine $f^* = \min_{(i,j) \in S_n} \{f_{(i,j)}\}$ and the machine m^* on which f^* could be realized.

Step 3. (1) Identify the operations set $(k, l) \in O$ such that $(k, l) \in S_n$, (k, l) require machine m^*, and $s_{(k,l)} < f^*$.
(2) Choose (i, j) from the set by a dispatching rule.
(3) Form PS_{n+1} by adding (i, j) to PS_n.
(4) Assign $s_{(i,j)}$ as the start time of (i, j).

Step 4. If a complete schedule has been generated, stop. Else, set $S_{n+1} = (i, j+1) \cup (S_n \backslash (i, j))$; if (i, j) is the last operation of job i, set $S_{n+1} = S_n \backslash (i, j)$; set $n = n+1$; and go to Step 2.

2.2. *Active Chains*

The *active chain* described in this paper is a variant of the active chain discussed by Giffler and Thompson [11]. For the description of active chains, we first introduce a concept called *next-follow* relation.

Definition

Let (i, j) denote the jth operation of job i. A *next-follow* relation is a relation between two operations (i, j) and (m, n) in a schedule such that (i, j) *next-follows* (m, n) if and only if: (1) the starting time of (i, j) is equal to the finishing time of (m, n), and (2) (m, n) is either the preceding operation of (i, j) of the same job, i.e. $(i, j-1)$, or the operation of a different job processed on the same machine as (i, j).

By definition, an operation could *next-follow* at most two other operations. The *next-follow* relation is illustrated in Fig. 1. The Gantt chart in Fig. 1 represents a schedule for a four-job, three-machine job shop, where i, j in each operation block stands for operation (i, j). For example, (2, 2) *next-follows* (3, 3), (3, 3) *next-follows* (4, 3), (4, 3) *next-follows* (4, 2), while (4, 2) *next-follows* two operations—(4, 1) and (3, 1).

Simply speaking, an *active chain* of an operation (i, j) is a set of operations that detemine the finishing time of (i, j) in a schedule. The formal definition of an active chain is the following.

Definition

An *active chain* of operation (i, j) is a set of operations including (i, j) and an operation without a predecessor, such that (1) each operation in the set, except the operation without a predecessor, *next-follows* exactly one operation in the set, (2) for each operation in the set, except (i, j), there is exactly one operation in the set that *next-follows* the operation.

An active chain of (i, j) is, therefore, a set of operations connected by the *next-follow* relations, starting with an operation without a predecessor (the leftmost block in a row of Gantt chart) and ending with (i, j). The length of an active chain is defined as the sum of the processing times of all operations in the chain. If the earliest starting times of all jobs are all zero, the finishing time of (i, j) will be simply the length of its active chain.

Four active chains are shown in the Gantt charts in Fig. 2. The four Gantt charts are the same schedule for a four-job, three-machine job shop as that shown in Fig. 1. The active chains of operation (3, 3), (1, 3), (2, 3) and (4, 3) are shown in Fig. 2(a), 2(b), 2(c) and 2(d), respectively, by the shaded operation blocks. The schedule in Fig. 2 is an active schedule, i.e. no operations in the schedule can be started earlier without delaying some other operations. This can be confirmed

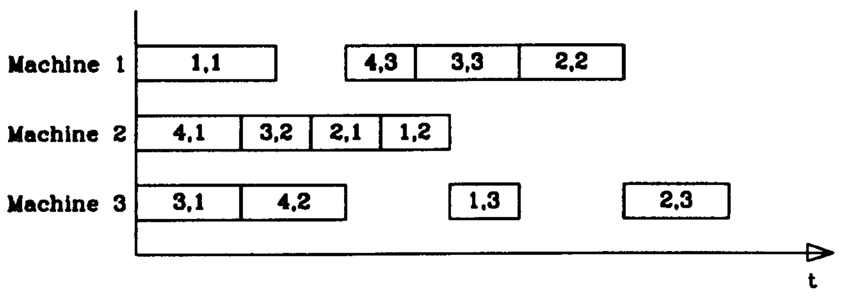


Fig. 1. An illustration of next-follow relations.

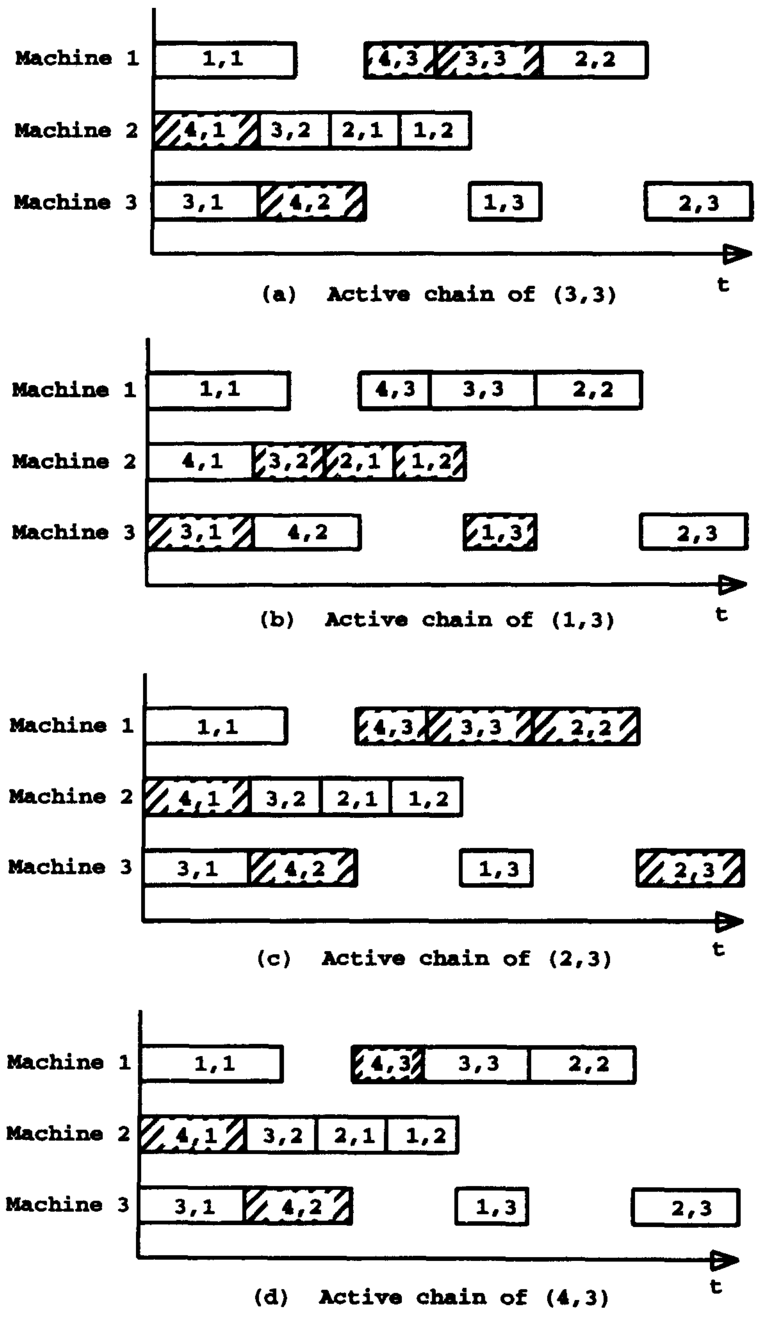


Fig. 2. An illustration of active chains.

simply by observation in this simple case. Note that an operation could have more than one active chain. For example, (1, 3) has two active chains: one includes (3, 1), (3, 2), 2, 1), (1, 2) and (1, 3) [illustrated by shaded operation block in Fig. 2(b)], another includes (4, 1), (3, 2), (2, 1), (1, 2) and (1, 3), because the finishing times of (3, 1) and (4, 1) are the same.

2.3. *Properties of Active Chains*

The scheduling algorithm development in this paper improves a schedule through an active chain manipulation scheme. The two lemmas below state (1) a relationship between active schedules and active chains, and (2) an intuitive yet important property of active chains. Together, they constitute the basis for the development of the active chain manipulation scheme (proofs are presented in Appendix).

Lemma 1

There exists at least one active chain for each operation in an active schedule.

Lemma 2

A necessary condition for an operation in an active schedule to be finished earlier is that at least one precedence relation in one of the operations's active chains is changed.

3. MAKE-SPAN REDUCTION SCHEDULING ALGORITHM

An iterative improvement algorithm for the make-span reduction scheduling problem will be described in this section. The algorithm searches an active schedule space through a series of active chain manipulations. Tabu search is used in the algorithm to avoid the trap of local optima.

3.1. Iterative Improvement Procedure

An iterative improvement procedure is a tuple ($\bar{S}$, $\bar{C}$, $\bar{T}$, $\bar{N}$, $\bar{R}$). The meaning of the components of the tuple are as follows: $\bar{S}$ is the set of all possible configurations or solutions of the problem; $\bar{C}$ is the *cost function* that defines the cost for each configuration; $\bar{T}$ is the *transition*, which is a set of operations that transform a configuration into another; $\bar{N}$ is the *neighborhood structure*; for each configuration s, $\bar{N}(s)$ is a subset of $\bar{S}$, called the neighborhood of s, that is the set of configurations that can be generated from s through one *transition*; $\bar{R}$ is the *transition rules*, which are rules used to choose a transition from all feasible transitions for a given configuration. The tuple ($\bar{S}$, $\bar{C}$, $\bar{T}$, $\bar{N}$, $\bar{R}$) of the iterative improvement procedure for the make-span reduction scheduling problem is described in more detail as follows.

3.2. Configuration

The configurations of the make-span scheduling problem are all the schedules such that (1) they are feasible, and (2) they are active. For convenience, the terms configuration and schedule shall be used interchangeably.

3.3. Cost Function

The cost of a schedule is the make-span of the schedule. If all jobs' available times are zero, the make-span is the length of the longest active chain in the schedule. The make-span can be calculated by simply adding up the processing times of all operations in the longest active chain. If some jobs' available times are greater than zero, by defining the time intervals between time zero and those non-zero available times as dummy operations, the make-span of a schedule can still be calculated as the length of the longest active chain in the schedule.

3.4. Transitions

3.4.1. Two-step transitions

A transition is a change of the machine processing sequences of some operations, so as to transform a schedule into another. The transitions in the scheduling algorithm are designed to work on active schedules and generate different active schedules. Let S denote an active job shop schedule. A *transition* is applied to S and consists of the following two steps:

Step 1. Make one move that changes at least one precedence relation between the operations in one of the longest active chains in S.

Step 2. Adjust machine processing sequences to make the resulting schedule active (the algorithm to carry out the adjustment is described in Section 3.7). We note here that every active schedule, by its definition, is feasible.

The first step in a transition is designed based on a simple fact: if the precedence relations between the operations in any of the longest active chains remain the same, the longest active chain's length, i.e. make-span, cannot be shortened (it is worth noting that the longest active chain is equivalent to the longest path in a disjunctive graph, see Balas [13] and Van Laarhoven *et al.* [7]). This fact is formally stated in the following theorem.

Theorem

A necessary condition for the make-span of an active schedule to be shortened is that it changes at least one precedence relation between the operations in one of the longest active chains in the active schedule.

Proof. By Lemma 2 and the relationship between make-span and the length of the longest chains. □

Although the number of the longest active chains in an active schedule could be more than one, only one longest active chain is identified at each of the iterative steps of the scheduling algorithm. This longest active chain is identified by a rule that if an operation (i, j) *next-follows* two operations, $(i, j-1)$ will be included in the active chain instead of the operation processed on the same machine with (i, j). For example, in Fig. 2(c), in the active chain of (2, 3), (4, 2) next-follows two operations: (4, 1) and (3, 1). By the rule, (4, 1) is included in the active chain of (2, 3), instead of (3, 1).

3.4.2. *Three types of moves*

Let O_A^k, where A is operation identifier, denote that operation O_A is in the kth position in the processing sequence on a machine, i.e. O_A is the kth operation processed on the machine. The move in Step 1 of the transition is chosen from the following three types of moves.

Pairwise exchange. Exchange the positions of two operations in a machine processing sequence. The two operations exchanged could be either adjacent or separated by one or more other operations. For instance, $O_A^iO_B^{i+1}O_C^{i+2} \Rightarrow O_B^iO_A^{i+1}O_C^{i+2}$ is a pairwise exchange of two adjacent operations O_A and O_B; $O_A^iO_B^{i+1}O_C^{i+2} \Rightarrow O_C^iO_B^{i+1}O_A^{i+2}$ is a pairwise exchange of two operations O_A and O_C, which are separated by operation O_B.

Move ahead. Move an operation ahead (i.e. left in a Gantt chart) n positions in a machine processing sequence. If an operation is moved ahead from the kth position to the $(k-n)$th position, $n \leqslant k$, in the processing sequence, the operations in the $(k-n)$th position to $(k-1)$th position before the move will be in the $(k-n+1)$th position to kth position after the move, i.e. the position of each of the operations will be incremented by one. For instance, $O_A^iO_B^{i+1}O_C^{i+2} \Rightarrow O_C^iO_A^{i+1}O_B^{i+2}$, if operation O_C is moved ahead two positions.

Move back. Move an operation back (i.e. right in a Gantt chart) n positions in a machine processing sequence—this move is symmetric to move ahead. If an operation is moved back from the kth position to the $(k+n)$th position in the processing sequence, the operations in the $(k+1)$th position to $(k+n)$th position before the move will be in the kth position to $(k+n-1)$th position after the move, i.e. the position of each of the operations will be decremented by one. For instance, $O_A^iO_B^{i+1}O_C^{i+2} \Rightarrow O_B^iO_C^{i+1}O_A^{i+2}$, if operation O_A is moved back two positions.

3.4.3. *Transition set*

A transition set is constructed at each of iterative steps. It contains all transitions such that: (1) each of their first steps is one of the above three moves, (2) the moves change at least one precedence relation between the operations in the active chain identified at the iterative step, and (3) the moves are not forbidden. The reader is referred to Section 3.9 for an example of the construction of transition sets.

3.5. *Neighborhood*

The scheduling algorithm is designed to search only an active schedule space. Given a schedule and the transitions defined above, the neighborhood of the schedule is therefore all active schedules that can be generated by transitions that change at least one precedence relation among operations in a longest active chain in the schedule.

3.6. *Rules*

There are two rules used in the scheduling algorithm to choose a transition from a transition set. One is a what-not-to-do rule, that is used to determine forbidden transitions. Another is a what-to-do rule used to choose a transition from the transitions that are not forbidden. The what-not-to-do rule is an implementation of Tabu search. The what-to-do rule is a greedy search rule.

3.6.1. Tabu search

Tabu search (see Glover [9, 10]) is a "higher level" heuristic procedure designed to guide other methods to escape the trap of local optimality. Tabu search can be used with any procedures that act on a set of moves to transform one solution to another. It has been successfully applied to many discrete optimization problems [10], e.g. some machine scheduling problems (see, Barnes and Laguna [14], Skorin-Kapov and Vakharia [15] and Laguna [16]). Tabu search is characterized by defining certain moves as forbidden moves, therefore "Tabu" moves. The forbidden moves are determined in Tabu search based on historical search information stored in a data structure called Tabu list. Typically, a move is forbidden if it reverses a recent move and therefore will potentially cause cycling. A Tabu list is actually a "what-not-do-do" list; it provides "memory" to a search procedure. This kind of "memory" can help the search procedure to diversify its search and avoid the traps of local optimality.

Note that the Tabu method can be distinguished from other types of search procedures by the way that how the search history is remembered. The Tabu method remembers only part of the search history. In contrast, optimization procedures, such as the branch and bound procedure and the A^* procedure, are designed to rigidly remember the entire search history. Furthermore, the simulated annealing procedure, on the other hand, does not remember the search history at all. In a sense, the Tabu method is a middle ground of the other two types of procedures.

The rule for determining the forbidden moves in the scheduling algorithm is embedded in the use of a Tabu list. In general, it is as follows: in the algorithm, the Tabu list is implemented as a finite queue consisting of **q** (the size of Tabu list) nodes. A FIFO queue discipline is employed. Each of the nodes in the Tabu list stores an operation identifier and a forbidden position of the operation in the processing sequence on a machine. The Tabu list is updated at each move in the search procedure. When an operation is moved from an old position to a new position in a processing sequence, the operation and its old position are added to the tail of the Tabu list, the node at the head of the Tabu list is removed. The operation-position pairs in the Tabu list represent the forbidden moves. As long as an operation-position pair is on the Tabu list, the operation is forbidden from being moved into the corresponding position. As a result, an operation cannot be moved back to a position in the processing sequence it left until after **q** moves.

In implementing the rule, however, there is a complicating factor: a transition may change the positions of two or more operations in the processing sequences. Typically, among the operations that are moved to new positions by the transition, some are not in forbidden positions, but others are. Two problems arise: (1) the determination of the forbidden transitions and (2) the rule of updating the Tabu list. These two problems are dealt with, in the algorithm, according to the move types in the first steps of transitions.

Pairwise exchange. A transition is not forbidden if and only if in the resulting schedule neither of the exchanged operations is in a forbidden position. If a transition with its first step as a pairwise exchange is chosen to be carried out, two nodes are added to the Tabu list after the transition—each node contains one of the two exchanged operations and its position in a machine processing sequence before the transition.

Move ahead (back). A transition is not forbidden if and only if in the resulting schedule the operation moved ahead (back) is not in a forbidden position. If a transition with its first step as a move ahead (back) is chosen to be carried out, one node is added to the Tabu list after the transition, which contains the identifier of the operation moved ahead (back) and its position in a machine processing sequence before the transition.

3.6.2. Greedy search

The what-to-do rule chooses the transition to be carried out, in a greedy manner, from all possible transitions that are not forbidden. Among all transitions that are not forbidden, a transition is chosen if it results in a schedule that has the smallest make-span. Let T denote the set of all transitions that are not forbidden; and MS_u the make-span of the schedule resulted from transition u. Transition t is chosen if $MS_t = \min_{u \in T}(MS_u)$.

3.7. Schedule Adjustment Algorithm

In Step 2 of a transition, adjustment is made to bring a schedule that is not active back to an active one. The algorithm to carry out this adjustment is an active schedule generation algorithm denoted as ASG2 Algorithm, which is a variant of ASG Algorithm described in Section 3 (with the only difference in Step 3). The ASG2 Algorithm is designed such that in adjusting a schedule, the machine processing sequences resulted from the move in Step 1 of a transition are changed as little as possible. In the following description of ASG2 Algorithm, the notations have the same meaning as those used in ASG Algorithm in Section 2.

ASG2 Algorithm

```
Step 1. Initialize $n=0$ and $PS_n=\varnothing$; $S_n$ is initialized to contain all operations without
        predecessors.
Step 2. Determine $f^*=\min_{(i,j)\in S_n}\{f_{(i,j)}\}$ and the machine $m^*$ on which $f^*$ could be
        realized.
Step 3. (1) Identify the operation set $(k,l)\in O$ such that $(k,l)\in S_n$, $(k,l)$ requires
            machine $m^*$, and $s_{(k,l)}<f^*$.
        (2) Choose $(i,j)$ such that among all operations in the operation set identified
            in (1), it is first processed on machine $m^*$ in the machine processing
            sequences resulting from the first step of the transition.
        (3) Form $PS_{n+1}$ by adding $(i,j)$ to $PS_n$.
        (4) Assign $s_{(i,j)}$ as the starting time of $(i,j)$.
Step 4. If a complete schedule has been generated, stop. Else, set $S_{n+1}=(i,j+1)\cup$
        $(S_n\backslash(i,j))$; if $(i,j)$ is the last operation of job $i$, set $S_{n+1}=S_n\backslash(i,j)$; set $n=n+1$;
        and go to Step 2.
```

3.8. Make-span Reduction Scheduling Algorithm

Let T denote the set of transitions such that the move in Step 1 of each transition is not forbidden (for efficiency, the forbidden moves are only checked in Step 1 of the transitions); S^* the schedule with the least make-span obtained so far; S^i the schedule generated at iterative step i; MS_u the make-span of the schedule resulted from transition u. The make-span reduction scheduling algorithm, denoted by ACM Algorithm (ACM—Active Chain Manipulation), is as follows.

ACM Algorithm

```
Step 1. Set $i=0$. Generate the initial active schedule $S^0$ using ASG Algorithm. Set
        $S^0$ to $S^*$.
Step 2. Identify the operations in a longest active chain of $S^i$; construct the transition
        set $T$ from the transitions that are not forbidden by looking up the Tabu list.
Step 3. Test all transitions in $T$ for their resulting schedules' makespan. Transition
        $t$ is chosen if: $MS_t=\min_{u\in T}(MS_u)$.
Step 4. Make transition $t$ to transform $S^i$ into $S^{i+1}$. Update the Tabu list. If the
        make-span of $S^{i+1}$ is less than that of $S^*$, set $S^{i+1}$ to $S^*$. If terminal condition
        is met, which could be a prescribed length of computation time, or a number
        of iterative steps, stop. Otherwise, set $i=i+1$ and go to Step 2.
```

The flow chart of ACM Algorithm is shown in Fig. 3, where S stands for a shop schedule; $ms(S)$ the make-span of S; S^* the best shop schedule obtained so far; S^n the shop schedule generated by the nth transition in a transition set.

3.9. An Example

An example is presented below to illustrate the functioning of the ACM algorithm. The job shop to be scheduled has four jobs and three machines. Each job has three operations that are processed on the three distinct machines. Let $p(i, j)$ denote the processing time of operation (i, j), $p(4, 3)$, $p(3, 2)$, $p(2, 1)$, $p(1, 2)$ and $p(1, 3)$ are two time units; $p(3, 3)$, $p(2, 2)$, $p(4, 1)$, $p(3, 1)$, $p(4, 2)$ and $p(2, 3)$ are three time units; and $p(1, 1)$ is four time unit.

The notations used in Figs 4–7 are as follows. The column under "MOVES" lists all possible

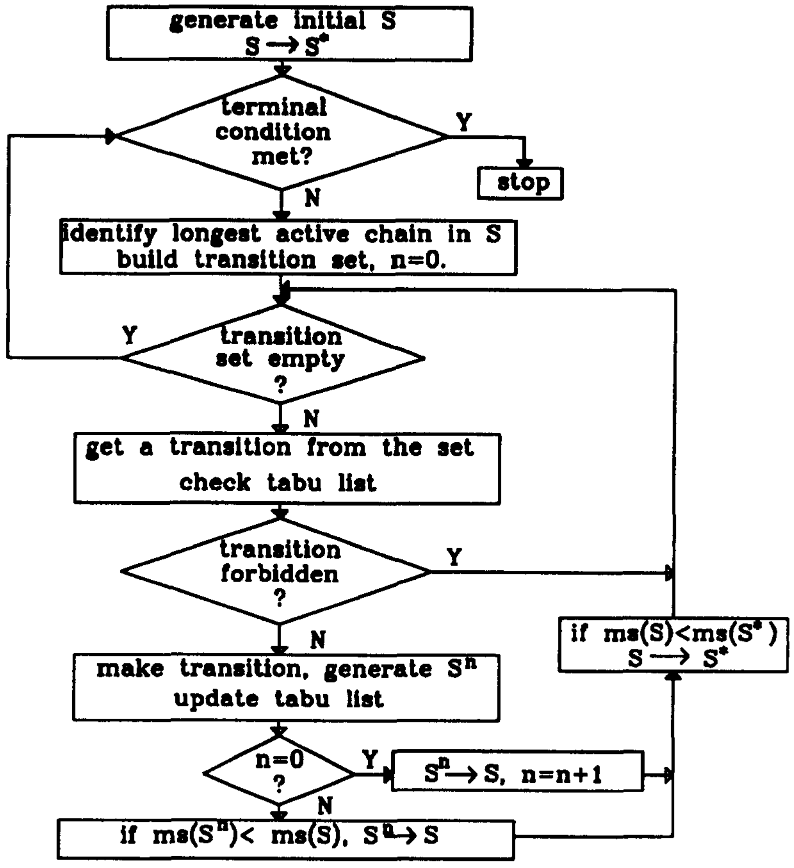
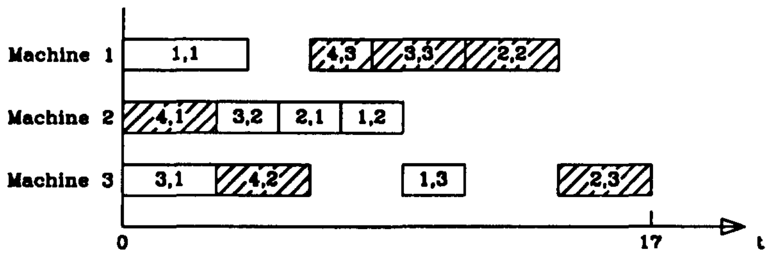


Fig. 3. Flow chart of ACM Algorithm.

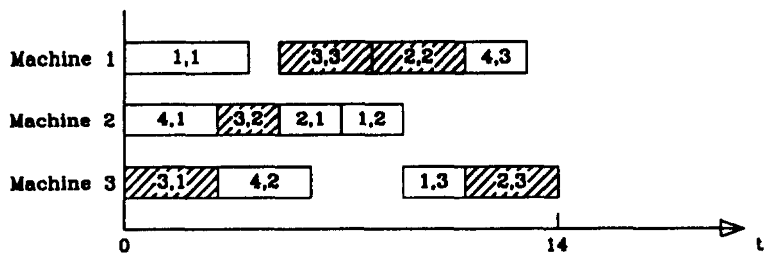


MOVES	STATUS	MAKE-SPAN	TABU-LIST
(3,3), (2,2) EX		15	empty
(4,3), (3,3) EX		16	
(4,3), (2,2) EX		15	
(4,3) M-B(2)		14	
(2,2) M-A(2)		15	

MOVE CHOSEN: (4,3) M-B(2)

Fig. 4. The original schedule of the example.

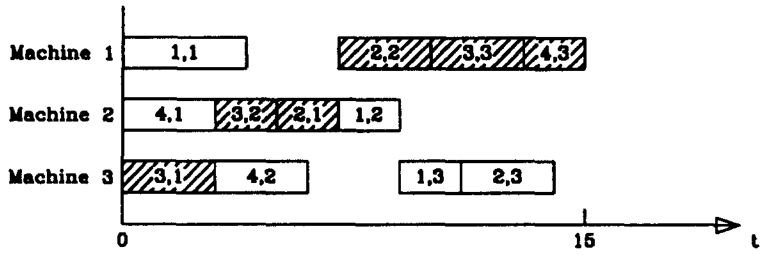
moves that change at least one precedence relation in a longest active chain in the schedule. The active chains are represented by shaded operation blocks in Gantt charts. The notation "(i, j), (m, n) EX", stands for pairwise exchange of (i, j) and (m, n). The labels "(i, j) M-B(2)" and "(i, j) M-A(2)" stand for (i, j) moving back two positions and moving ahead two positions, respectively. The column under "STATUS" lists the moves' forbidden status. If a move is forbidden, it is labeled "TABU". All listed moves without the label "TABU" constitute the transition set T of the associated iterative step. The column under "MAKE SPAN" lists the make-spans of the schedules that result from the moves listed in the same row in the first column. The column under "TABU-LIST" is the content



MOVES	STATUS	MAKE-SPAN	TABU-LIST
(3,3), (2,2) EX		15	(4,3) 2
(3,3), (4,3) EX	TABU		
(3,3) M-B(2)		15	
(4,3) M-A(2)	TABU		

MOVE CHOSEN: (3,3), (2,2) EX

Fig. 5. Schedule after one transition from the schedule in Fig. 4.



MOVES	STATUS	MAKE-SPAN	TABU-LIST
(3,3), (4,3) EX		15	(4,3) 2
(2,2), (3,3) EX	TABU		(3,3) 2
(3,2), (2,1) EX		14	(2,2) 3
(3,2), (1,2) EX		15	
(2,2), (4,3) EX	TABU		
(4,1), (2,1) EX		17	
(2,2) M-B(2)		17	
(3,2) M-B(2)		14	
(2,1) M-A(2)		14	
(4,3) M-A(2)	TABU		

MOVE CHOSEN: (3,2), (2,1) EX

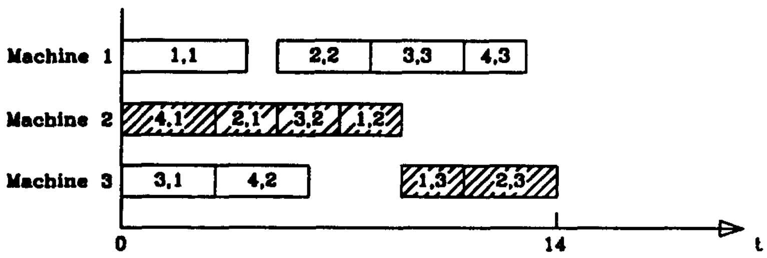
Fig. 6. Schedule after two transitions from the schedule in Fig. 4.

of the Tabu list of the associate iterative step. Each row of the column represents a node in the Tabu list. The notation "(*i*, *j*) *n*" is the operation identifier and position pair, which means that (*i*, *j*) is not allowed to enter the *n*th position in a processing sequence. The make-span of a schedule is marked along the time axis.

Four transitions and resulting schedules are illustrated in Figs 4–8. In implementing the ACM algorithm, the pairwise exchanges used are the pairwise exchange of adjacent operations and the pairwise exchange of two operations separated by another one; and the move ahead (back) is limited to moving ahead (back) two positions. The initial schedule is illustrated in Fig. 4. The Tabu list is assumed empty at the beginning. In choosing a move, ties in make-span are broken arbitrarily.

4. COMPARISON WITH DISJUNCTIVE GRAPH-BASED PARIWISE EXCHANGE

Job shop schedule improvement by pairwise exchanges of operations has been previously studied based on a disjunctive graph schedule representation (see, for example, Balas [13], Van Laarhoven *et al.* [7]). In those studies, however, the pairwise exchanges are only applied to adjacent operations, which is implemented as reversing the direction of a disjunctive arc in a disjunctive graph, and the pairwise exchanges are not related to active schedule generation.



MOVES	STATUS	MAKE-SPAN	TABU-LIST
(1,3), (2,3) EX		13	(4,3) 2
(3,2), (1,2) EX		14	(3,3) 2
(4,1), (2,1) EX		14	(2,2) 3
(2,1), (3,2) EX	TABU		(2,1) 3
(2,1), (1,2) EX		16	(3,2) 2
(4,1), (3,2) EX		16	
(4,1) M-B(2)		18	
(2,1) M-B(2)		15	
(1,2) M-A(2)		14	
(3,2) M-A(2)		15	

MOVE CHOSEN: (1,3), (2,3) EX

Fig. 7. Schedule after three transitions from the schedule in Fig. 4.

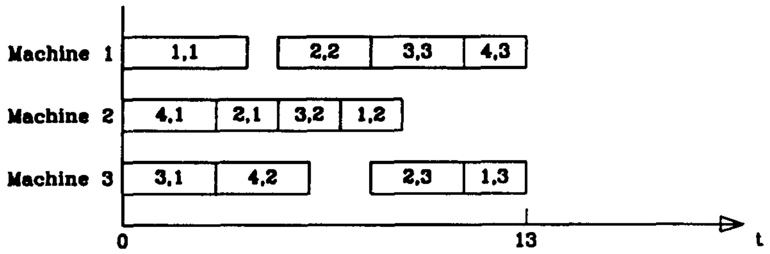


Fig. 8. Schedule after four transitions from the schedule in Fig. 4.

Compared with the disjunctive graph-based pairwise exchanges, the transitions described in this paper have the following desirable features.

Fuller and more efficient exploration of alternative schedules

Let P-E denote a pairwise exchange of adjacent operations, and let T-R be the set of transitions described in this paper. T-R includes more move types than P-E does. As a result, the neighborhood of a schedule can be more fully explored by T-R. Search efficiency can also benefit from more move types. For example, one pairwise exchange of two operations that are separated by n other operations requires $2n+1$ P-Es to accomplish.

Better evaluation of moves

It is desirable that if a move is "correct", the make-span in the resultant schedule should show improvement. However, it could be inappropriate to evaluate a P-E only on the make-span of the resultant schedule. This is because a P-E may result in a schedule that is not active. When a schedule is not active, its make-span usually can be shortened by a schedule adjustment that results in an active schedule. The transition in this paper combines a move and such a schedule ajustment so that the move can be more adequately evaluated.

5. COMPUTATIONAL RESULTS

The ACM Algorithm has been tested on N-job, M-machine job shop scheduling problems in which each job is processed on each machine once and only once. The test set includes 26 benchmark problems including: (1) a 10-job, 10-machine problem due to Muth and Thompson [6], which is

Table 1. Computational results of 10-job 10-machine make-span problems

	SB Procedure		ACM Algorithm			
Problem	Make-span	T(s)	Make-span	T(s)	*Make-span**	*T*(s)*
M-T	930*	851	—	—	930*	157
LA16	978	240	975	13	—	—
LA17	787	192	786	38	784*	125
LA18	859	225	853	12	848*	64
LA19	860	240	850	19	842*	51
LA20	914	289	909	18	902*	190

Table 2. Computational results of 16-job 10-machine make-span problems

	SB Procedure		ACM Algorithm			
Problem	Make-span	T(s)	Make-span	T(s)	*Make-span**	*T*(s)*
LA21	1084	362	1074	28	—	—
LA22	944	419	941	278	—	—
LA23	1032*	225	—	—	1032*	60
LA24	976	434	954	61	—	—
LA25	1017	430	1010	32	—	—

Table 3. Computational results of 20-job 10-machine make-span problems

	SB Procedure		ACM Algorithm			
Problem	Make-span	T(s)	Make-span	T(s)	*Make-span**	*T*(s)*
LA26	1224	744	—	—	1218*	280
LA27	1291	837	1277	198	—	—
LA28	1250	901	1245	507	—	—
LA29	1239	892	1234	107	—	—
LA30	1355*	551	—	—	1355*	389

Table 4. Computational results of 30-job 10-machine make-span problems

	SB Procedure		ACM Algorithm			
Problem	Make-span	T(s)	Make-span	T(s)	*Make-span**	*T*(s)*
LA31	1784*	38.3	—	—	1784*	38
LA32	1850*	39.1	—	—	1850*	10
LA33	1719*	25.6	—	—	1719*	94
LA34	1721*	27.6	—	—	1721*	154
LA35	1888*	21.3	—	—	1888*	148

denoted by M-T in Table 1, and (2) 25 other problems which are problems 16–40 from Lawrence [17]—these problems are denoted in the following tables as LA16 to LA40. The optimal solutions to most of the problems are known. The test results are listed in Table 1–5. For reference, the computational results of the *Shifting Bottleneck Procedure* [8] for the same set of problems are also included in the tables.

In testing the ACM Algorithm, the initial schedules are generated by a dispatching rule that chooses the jobs to be scheduled as the ones with the most remaining work content. The following were observed in our empirical testing: (1) when the size of the Tabu list is too small, cycling frequently occurs; (2) a very large Tabu list size slows the schedule's improvement process, as it imposes unnecessary restrictions to the search procedure; and (3) when the Tabu list' sizes are in the range of 1/8–1/5 of the number of operations in the problems, the ACM Algorithm consistently generates better schedules—the results listed in Tables 1–5 are generated with the Tabu list size in this range.

In the tables, make-spans and the CPU times consumed in generating the schedule with the make-spans are listed side-by-side. The term *SB procedure* stands for the shifting bottleneck

Table 5. Computational results of 15-job 15-machine make-span problems

	SB Procedure		ACM Algorithm			
Problem	Make-span	T(s)	Make-span	T(s)	*Make-span**	*T*(s)*
LA36	1305	735	1303	71	—	—
LA37	1423	837	1422	93	—	—
LA38	1255	1079	1245	475	—	—
LA39	1273	669	1269	378	—	—
LA4	1269	899	1255	213	—	—

procedure; T(s) the CPU time in seconds. The two columns under *SB procedure* are the computational results directly from Adams *et al.* [8]. The four columns under ACM Algorithm are the computational results generated by the ACM Algorithm. The test is conducted as follows: if the optimal make-span value of a test problem was reached by the shifting bottleneck procedure, the ACM Algorithm is run on this problem until the optimal make-span value is reached, no matter how long it takes. If the optimal make-span value of a test problem was not reached by the shifting bottleneck procedure, the ACM Algorithm will be run on this problem for the same CPU time length as that consumed by the shifting bottleneck procedure. If during this CPU time length, the optimal make-span value is reached by the ACM Algorithm, the optimal value and the CPU time spent to generate the optimal value are listed in the last two columns under ACM Algorithm. Otherwise, the cpu time consumed to generate a make-span value, which is better than that generated by shifting bottleneck procedure, is listed along with the value in the first two columns under ACM Algorithm. *Make-span** and *T*(s)* stand for optimal make-span value and the CPU time consumed by the ACM Algorithm to generate the optimal make-span value. If a make-span value is optimal, it is labeled with "*". The notation "-" is used to represent: (1) ignorable intermediate result, or (2) unachieved optimal value.

The shifting bottleneck procedure was tested on Vax 11/780 by Adams *et al.* [8]. Limited by the unavailability of the Vax machine, the ACM Algorithm is tested on a Sun Sparcstation IPC. The computational speed of the Sun Sparcstation is roughly four times as fast as that of Vax 11/780. Taking the speed difference into account, the CPU times of the ACM Algorithm listed are equivalent Vax 11/780 computation times, which are calculated by multiplying the CPU times used on the Sun Sparcstation by four, i.e. the actual CPU times of the Sun Sparcstation used are 1/4 of those listed. The CPU times of the shifting bottleneck procedure listed are actual Vax 11/780's CPU time.

From the computational results, the ACM Algorithm appears to be effective. The optimal schedules have been generated for half of the test problems though optimality is not addressed by the ACM Algorithm. For most of the test problems, the ACM Algorithm has generated better schedules in shorter time than the shifting bottleneck procedure did.

6. CONCLUDING REMARKS

A make-span reduction scheduling algorithm was developed in this paper. The basic method employed in the algorithm is based on searching an active schedule space through active chain manipulation using Tabu search technique. This method itself does not depend on the special characteristics of make-span scheduling problem, therefore, it has the potential to be adapted to dealing with other types of job shop scheduling problems, such as flow reduction problems and tardy penalty cost reduction problems.

Central to the transitions in the scheduling algorithm is the generation of active schedules, which is also the key to implementing dispatching rules. This makes the active chain manipulation-based algorithm and dispatching rules closely related. This relationship may provide a basis for constructing hydrid scheduling systems that are capable of integrating search procedures and dispatching rules. The hybrid systems can be designed such that search procedures are implemented whenever possible for effectiveness, and the dispatching rules are carried out when quick responses are necessary and to accommodate the lack of predictive information in uncertain manufacturing environments.

The search procedure developed in this paper is opportunistic in nature—as the moves are made only based on local improvements. Although the alternative schedule space is not explored in a systematic way, the search procedure is still able to generate very good schedules. This may be considered as an indicator of the potential of iterative improvement-based algorithms in dealing with computationally difficult problems. With increasing computing power, the iterative improvement approach may be considered as a practical alternative to less effective one-pass heuristics for a variety of manufacturing decision problems. The key to the performance of iterative improvement procedures seems to be the full exploitation of the structure of the problems to be solved.

Acknowledgement—The authors would like to thank two anonymous referees for their helpful comments.

REFERENCES

1. B. J. Lageweg, J. K. Lenstra and A. H. G. Rinnooy Kan, Job-shop scheduling by implicit enumeration. *Mgmt Sci.* **24**, 441–450 (1977).
2. J. Carlier and E. Pinson, An algorithm for solving the job-shop problem. *Mgmt Sci.* **35**, 164–176 (1989).
3. D. Applegate and W. Cook, A computation study of the job-shop scheduling problems. *ORSA Jl Comput.* **3**, 149–156 (1991).
4. J. H. Blackstone, D. T. Phillips and G. L. Hogg, A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *Prod. Res.* **20**, 27–45 (1982).
5. S. D. Wu and R. A. Wysk, An application of discrete-event simulation to on-line control and scheduling in flexible manufacturing systems. *Prod. Res.* **27**, 1603–1623 (1989).
6. J. F. Muth and G. L. Thompson (Editors), *Industrial Scheduling.* Prentice-Hall, Englewood Cliffs, NJ (1963).
7. P. J. M. Van Laarhoven, E. H. L. Aarts and J. K. Lenstra, Job shop scheduling by simulated annealing. *Ops Res.* **8**, 487–503 (1992).
8. J. Adams, E. Balas and D. Zawack, The shifting bottleneck procedure for job shop scheduling. *Mgmt Sci.* **34**, 391–401 (1988).
9. F. Glover, Future paths for integer programming and links to artificial intelligence. *Computers Ops Res.* **13**, 533–549 (1986).
10. F. Glover, Tabu search, Part I. *ORSA Jl Comput.* **1**, 190–206 (1989).
11. B. Giffler and G. L. Thompson, Algorithm for solving production scheduling problems. *Ops Res.* **8**, 487–503 (1960).
12. K. R. Baker, *Introduction to Sequencing and Scheduling.* John Wiley, New York (1974).
13. E. Balas, Machine sequencing via disjunctive graphs: an implicit enumeration algorithm. *Ops Res.* **17**, 941–957 (1969).
14. J. W. Barnes and M. Laguna, Solving the multi-machine weighted flow time problem using Tabu search. *IIE Trans.* **25**, 121–128 (1993).
15. J. Skorin-Kapov and A. J. Vakharia, Scheduling a flow-line manufacturing cell: a tabu search approach. *Prod. Res.* **31**, 1721–1734 (1993).
16. M. Laguna, The application of heuristic methods and artificial intelligence to a class of Producting scheduling problems. Unpublished Ph.D. Dissertation. The University of Texas at Austin (1990).
17. S. Lawrence, Resource restrained project scheduling: an experimental investigation of heuristic scheduling techniques. GSIA, Carnegie Mellon University (1984).
18. R. W. Conway, W. L. Maxwell and L. W. Miller, *Theory of Scheduling.* Addison-Wesley, Reading, MA (1967).
19. D. Sun, Dynamic job shop scheduling—an integrated approach. Ph.D. dissertation. SUNY at Buffalo (1993).

APPENDIX

Proofs

Lemma 1

There exists at least one active chain for each operation in an active schedule.

Proof (by construction). In an active schedule, any operation with a predecessor operation *next-follows* at least one operation. Otherwise, the operation can be started earlier without delaying any other operations, but this will contradict the definition of an active schedule. There are two situations: (1) the operation has a predecessor, and (2) the operation does not have a predecessor. If the operation does not have a predecessor, the active chain is the operation itself. Otherwise, the active chain of an operation can be generated as follows. Add one operation at a time to the chain, starting from a given operation and selecting the next operation by *next-follow* relation; as mentioned above, for an operations with a predecessor operation, such operation can always be found. If an operation *next-follows* two operations, choose one arbitrarily; in this case, more than one active chain can be generated. This process continues until the operation without a predecessor is added to the set. □

Lemma 2

A necessary condition for an operation in an active schedule to be finished earlier is that at least one precedence relation among operations in one of the operation's active chains is changed.

Proof. Let an active chain of the operation be called A. By Lemma 1, there exists at least one active chain for each operation in an active schedule. In an active chain of an operation, each operation except the operation without a predecessor in A *next-follows* another operation that is either an operation processed on the same machine with the operation, or the operation's precedent operation of the same job. In both cases, the earlier start of the operation is blocked by an operation it *next-follows*, and the operation without predecessor is blocked by its own available time. If the precedence relations among the operations in A is not changed, the *next-follow* relations will remain the same for all operation pairs; therefore, no operation in A can be started earlier. As a result, the operation cannot be finished earlier. □