Combining Heuristics and Integer Programming for Optimizing Job Shop Scheduling Models

by

Michael Yi Xin Chu

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Operations Research

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 1995

Author .....
Department of Electrical Engineering and Computer Science
January 18, 1995

Certified by .....
Jeremy F. Shapiro
Professor of Operations Research and Management
Thesis Supervisor

Accepted by .....
Thomas L. Magnanti
Co-director and Professor of Management Science, Operations Research Center

Combining Heuristics and Integer Programming for Optimizing Job Shop Scheduling Models

by

Michael Yi Xin Chu

Submitted to the Department of Electrical Engineering and Computer Science
on January 18, 1995, in partial fulfillment of the
requirements for the degree of
Master of Science in Operations Research

Abstract

In this thesis, we review and synthesize a number of algorithmic approaches for optimizing the job-shop scheduling problem (JSSP). The complexity of the JSSP leads to large and difficult combinatorial optimization models. In Chapter 1, we review first an analytical method based on Mixed Integer Programming (MIP). Second, we review an assumption-based heuristic method. In Chapter 2, the JSSP is solved by the Branch and Bound algorithm for MIP using the CPLEX^{TM} Callable Library. Experiments with the Branch and Bound search parameters are discussed. In Chapter 3, we review first Benders' Decomposition for MIP and the JSSP. Then, we discuss an approach combining Benders' Decomposition and the assumption-based heuristic. Experimental results with that approach are discussed. In Chapter 4, we discuss areas of future research and computational experimentation with the MIP and heuristic method for the JSSP.

Thesis Supervisor: Jeremy F. Shapiro
Title: Professor of Operations Research and Management

Acknowledgments

It is my great pleasure to acknowledge the many people who contributed to the completion of this work. First and foremost, I would like to express my sincere gratitude to my thesis supervisor, Professor **Jeremy F. Shapiro**, for his invaluable counsel, constant support and encouragement. Especially, I thank him for his confidence in me. He always had a positive word to say –– even during the difficult times. His standard and enthusiasm for excellence in scientific research will guide me always.

I would like to thank Professor **Richard C. Larson** for his time listening to my presentation on "The Visual Stress-Maximized Stripe and Gestalt" just one week before Christmas' 92. He opened my eyes to the fact that operations research can have very individual and very human characteristics. I would also like to thank Professor **Thomas L. Magnanti** for his excellent course on mathematical programming. The course gave me the basic background to complete this thesis. I would like to thank Professor **John N. Tsitsiklis** for his standard for excellence in scientific research. My weekly presentations on Neural Networks to him and Professor **Munther A. Dahleh** during my first year at MIT have still been one of my best academic experiences. I would like to thank Professor **Harold Abelson** and Professor **Gerald J. Sussman** for their enthusiasm in teaching 6.001. They are among the greatest educators. I would like to thank Professor **Vernon M. Ingram** and **Mrs. Ingram** for their advice, support and encouragement. They have made my life in Ashdown House and at MIT more joyful. I would like to thank **Paulette**, **Laura**, and **Cheryl** for keeping the Operations Research Center functioning smoothly and for providing such an excellent learning environment for me.

I would like to thank all who helped me go through difficult times at MIT. In particular, I would like to thank **Yue Fang** and his wife for their unforgettable friendship, encouragement, and an especially wonderful Thanksgiving' 94 dinner; **Chung-Piaw**

Teo for his invaluable help and advice on my research, which sometimes included suggestions that were so detailed that our conversation over the phone lasted for hours; and **Edieal Pinker**, my first MIT student buddy, for teaching me Markov Processes one Saturday morning during my first semester at MIT. Furthermore, I am indebted to **Rob Shumsky**, **David Markowitz**, **Zhihang Chi**, and **S. Raghavan** for their help and advice on many computer software and hardware problems.

I am grateful to my friends **Edcon Chang** from MIT Chemistry Department, **Andrew Fung** from MIT Aeronautics and Astronautics Department, and **David Zhang** from MIT Ocean Engineering Department for the many joyful hours spent outside MIT. I sincerely hope that our friendship will extend far beyond our departures from MIT.

I am grateful to my friend Dr. **Thomas P. Caruso**, who graduated from Sloan School almost 11 years ago. He taught me how to use spreadsheets and Lotus Notes. He counseled me about my career decisions, and he is and always will be my mentor.

My deepest appreciation goes to my family and relatives. I would like to thank my parents, **Kuang Wu** and **Jia Gao**, my sister **Yvonne**, and my brother **Philip** for their caring and love. I would like to thank my granduncle and grandaunt, **Pao San** and **Tsu Hui**, my three uncles, **Lawrence**, **Philip** and **JiaLi**, and their wives, for their love, trust and support. Without their unconditional love, support and encouragement, this thesis could not have been completed.

I am deeply grateful to **Sarah Britt** for her encouragement, caring, and love, expressed in her many long letters and phone calls from the west coast. All of those have been a constant source of energy.

Finally, I dedicate this thesis in memory of **Marion Stemper** from Karlsruhe University, Karlsruhe, Germany.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Job-Shop Scheduling

A job-shop scheduling problem (JSSP) involves jobs consisting of a variety of machine operations to be processed on a number of machines. For a classical $n-$job $m-$machine job shop problem, in which each job has exactly one operation on each machine, each machine must process n operations. For example, for a $4-$job $3-$machine JSSP, there are 4 jobs, each consisting of 3 tasks, which are processed on 3 machines, each in some sequence. The addition of a scheduling criterion, such as the objective of constructing a schedule of minimum length (makespan), completes the problem statement. The criterion of minimum length is assumed throughout this paper.

This problem is a deterministic, static JSSP. By deterministic we mean that the processing time, and sequence of each task are known exactly. By static we mean that all jobs to be scheduled are known at the start of the time period in question, and that no jobs will arrive (or be deleted) from the schedule once it is formulated.

A schedule for the $4-$job $3-$machine problem consists of 3 permutations of the 4 jobs. Each permutation gives the processing sequence of jobs on a particular ma-

chine. Now there are $4! = 24$ different permutations of 4 objects and, since each of the 3 permutations may be chosen independently from the rest, it follows that the total number of schedules is $(4!)^3 = 13824$. There are an infinite number of feasible schedules for any JSSP because an arbitrary amount of idle time can be inserted at any machine between adjacent pairs of operations.

Here we have a very small problem: only 4 jobs and 3 machines. Yet the number of possible contenders for the solution is very large. To be fair, we might be able to check through the 13824 possibilities in a short time on a fast computer. If 1000 schedules were checked each second (which would be very fast), the computer would solve the problem in just over 0.2 minutes. But suppose that it were a 5−job 4−machine problem. The number of schedules would now be $(5!)^4 = 2.1 \times 10^8$ and the computer would take over 57 hours to solve this new problem!

Thus, the complexity of job-shop scheduling leads to large and difficult combinatorial optimization models. Many approaches to the problem have been suggested in the literature. Research has focused in two general areas, analytical (and usually computationally intense) methods guaranteed to find optimal solutions and heuristic (and usually computationally simple) methods to find good solutions quickly. Analytical methods break down for large problem. The thesis focuses on combining analytical and heuristic methods to solve the JSSP.

Very small problems can be solved by using brute force to enumerate all the possible sequences. Slightly larger problems can be solved by enumerating the dominant members of equivalence classes of sequences. Erschler et al. (1983) take a "pyramidal" approach specific to single machine problems. Early research by Akers and Friedman (1955) discusses how to reduce the set of schedules using only non-numerical means. Giffler and Thompson (1960) exploit dominance by enumerating all of the members of the set of so-called Active Schedules. The defining characteristic of an

Active Schedules is smaller than the related set of Non-Delay Schedules. Although the size of the set of schedules is reduced, it is still not small enough to allow practical enumeration of a moderate size problem. As an example, Giffler et al. (1963) show that a simple problem consisting of 6 jobs on 6 machines, with each job consisting of 5 tasks, has 84802 active, feasible schedules.

More recent work with enumeration has used Branch and Bound. The objective here is to prune the enumeration tree in order to reduce the number of schedules tested. At each node in the tree, a lower bound on the objective (e.g. length) is calculated. If the lower bound associated with a given parent node is greater than the completion time of the best schedule found so far, the node is pruned from the tree, since the lengths of all tip nodes associated with the parent will be, at best, equal to the lower bound. Lageweg et al. (1977) refer to their work with Branch and Bound as "implicit enumeration," and use a disjunctive graph representation of the problem. We will give an example in the next section.

As we discussed in the last section, optimal JSSP solution procedures typically rely on enumeration methods such as Branch and Bound. While much progress has been made in this approach, the JSSP has proven to be particularly difficult. As a result of this inherent intractability, heuristic solution procedures are an attractive alternative.

Heuristic solution procedures for the JSSP also have been widely studied. Most heuristics for the classical JSSP are local search methods. The local search starts with an initial solution. It then searches for neighbors which are solutions generated by a small change from the present solution and substitutes the solution. The local search repeats this procedure. It is called an iterative improvement method or hill-climbing method when the substitution of the solution is allowed only where the objective function improves. Iterative improvement has weak points, however, because, as the

problem size grows, the number of neighbors increases and the efficiency of the search in finding a better solution deteriorates.

Second, a local search heuristic can become trapped at a local optimal solution having no better neighbor. These points depend on the definition of neighbors. For efficiency, the number of neighbors should be small, but this makes the number of local optimal solutions large. A resurgence of interest in local search has occurred in recent years due to the development of probabilistic local search methods such as simulated annealing, genetic algorithms, and Tabu search. In simulated annealing (Kirkpatrick 1983), substitution in the direction of inferior objective function values is allowed with some probability. By controlling the probability, the method escapes from a local optimal solution. Its weak point is that it can converge slowly. Trying to speed it up can cause the method to be trapped at a local optimal problem. In short, simulated annealing essentially has the same weak points as iterative improvement.

An additional important heuristic is the "Shifting Bottleneck" (SB) heuristic of Adams et al. (1988) for the JSSP with makespan objective. It sequences the machines one by one, successively, taking each time the machine identified as a bottleneck among the machines not yet sequenced. Every time after a new machine is sequenced, all previously established sequences are locally reoptimized. Both the bottleneck identification and the local reoptimization procedures are based on repeatedly solving certain one-machine scheduling problems. This heuristic has been shown to achieve excellent results.

Mixed Integer Linear Programming (MIP) formulations and Branch and Bound methods for the classical JSSP, which will be discussed in the next section, have traditionally been used to find analytical solutions. An assumption-based heuristic method for the classical JSSP will be discussed in Section 1.3.

1.2 MIP Formulations and Branch and Bound

Since 1959 several models have been proposed to solve the classical JSSP by MIP. Efficiency in solving those models differs according to the method of expressing the constraints, so that every machine can process at most one job at a time, and so on, and by the way of defining the $0-1$ integer variables. However, since the number of constraints and/or variables becomes very large even for small size problem in any formulation by MIP, those attempts are not very effective.

The performance of Branch and Bound in optimizing these MIP models depends heavily upon the strength of the lower bound used to estimate the cost of completing a partial solution. If the bound is exact (not an estimate) then an optimal solution can be found directly. If the lower bound is inexact then some barren branches will be explored even though they do not lead to an optimal solution. If the bound used is not a lower bound but an approximation accurate to some known accuracy, then an approximately optimal solution (within this known accuracy) can be found. Surveys of categorization and complexity of algorithms and problems over various objective criteria can be found in Graves et al. (1993).

JSSPs involve jobs consisting of a variety of machine operations to be processed on a number of machines. The complexity of job-shop scheduling leads to large and difficult combinatorial optimization models. Thus, the practical use of models and associated analytical and heuristic methods should be to identify demonstrably good schedules, rather than to persist in trying to find an optimal solution.

We consider a combinatorial optimization model proposed by Lageweg, Lenstra and Rinnooy Kan (1977) and a number of other authors for a large class of JSSPs. This model is comprised of a number of jobs, each consisting of a number of operations to be processed on preassigned machines. The objective is to minimize the total length of time required to finish all jobs. In so doing, the model must simultaneously

sequence the processing of operation assigned to each machine and ensure that the precedence relations among operations of each job are obeyed (Shapiro 1993).

We define the indices and index sets of the model as the following:

i = jobs, $i = 1, ..., I$,

n_i = number of operations in job i,

j = operations, $j = 1, \ldots, N = \sum_{i=1}^{I} n_i$,

k = machine, $k = 1, ..., K$,

k_j = machine on which operation j is to be performed,

$J_k = \{j | k_j = k\}$ = operations assigned to machine k,

$R_k = |J_k|$ = number of jobs assigned to machine k,

r = machine sequence order, $r = 1, ..., R_k$.

The assumption for this class of JSSPs is that the operations in a given job are to be processed sequentially; that is, operation $j-1$ must be completed before operation j may begin. Thus, there is a total ordering of the operations of each job. For notation purposes, we assume that the operations of job i are indexed by $j = N_{i-1} + 1, ..., N_i$, where

$$N_i = \sum_{g=1}^{i} n_g$$

Parameters

p_j = processing time for operation j,

T = upper bound on total processing time.

Variables

t_j = start time for operation j,

$$x_{jg} = \begin{cases} 1 & \text{if operation } j \text{ performed before operation } g \\ 0 & \text{if operation } g \text{ performed before operation} \Pi\ j \end{cases} \quad \text{for } j,\ g \in J_k$$

T = total processing time for all jobs.

Job-shop scheduling model

$$v = \min F \tag{1.1}$$

subject to

for $i = 1, ..., I$

$$t_j \geq t_{j-1} + p_{j-1} \quad \text{for } j = N_{i-1} + 2, \ldots, N_i, \tag{1.2}$$

$$F \geq t_{N_i} + p_{N_i}, \tag{1.3}$$

for $k = 1, ..., K$

$$\begin{matrix} t_g \geq t_j + p_j x_{jg} - T(1 - x_{jg}) \\ t_j \geq t_g + p_g(1 - x_{jg}) - T x_{jg} \end{matrix} \quad \text{for all } j,\ g \in J_k, \text{ and } j < g. \tag{1.4}$$

$$x_{jg} = 0 \text{ or } 1 \quad \text{for all } j,\ g \in K_j, \tag{1.5}$$

$$F \geq 0, \quad t_j \geq 0 \qquad \text{for } j = 1, ..., N. \tag{1.6}$$

The objective function (1.1) is to minimize the time to complete all jobs. The variable time F is, by (1.3), equal to the maximum of the completion times of the I jobs. The constraints (1.2) state that the start times of operation j for $j = 2$ through $j = n_i$ for each job i must occur after operation $j - 1$ has been completed. The start time for operation 1 is not constrained by this total precedence ordering.

The start times t_j of operations are also constrained by the sequence of operations to be performed on each machine k. Specifically, for each pair of operations j, g assigned to machine k, the constraints (1.4) state that either operation j will precede operation g, or that operation g will precede operation j. For example, if $x_{jg} = 1$, then the first of two constraints is binding ($t_g \geq t_j + p_j$), whereas if $x_{jg} = 0$, then the second of the two constraints is binding ($t_j \geq t_g + p_g$).

To sum up, the start time t_j of operation j is constrained both by the total precedence ordering on the operations of its job, and by the sequencing order of operations from a variety of jobs on the machine k_j on which j is processed. The

former constraints (1.2) and (1.3) are the simple constraints of a network optimization type found in critical path method models (Schrage, 1986). The latter constraints (1.4) are logical constraints of a type referred to in the literature as disjunctive (Roy & Sussmann, 1964; Balas, 1979).

The result is an MIP model of great size and complexity. If $I = 10$, $n_i = 10$ for all i, $K = 8$, and $R_k = 25$ for all k, the model would have 2580 constraints, 201 continuous variables, and 240 binary variables. To try to circumvent extreme computational difficulties, several researchers have proposed solution techniques based on combinations of Branch and Bound schemes, heuristics and lower bounds based on easy to optimize relaxations of the job scheduling model (Lageweg, Lenstra & Rinnooy Kan, 1977; Adams, Balas & Zawack, 1988).

A central construct is the disjunctive graph which consists of a node for each operation j with an associated weight p_j, plus a dummy node 0 for the start of all operations, and a dummy node $N + 1$ for the completion of all operations. For each consecutive pair of operations $j - 1$ and j of the same job, there is a directed conjunctive arc. For each pair of operations assigned to the same machine, there is a (undirected) disjunctive edge. The major task in optimizing a given JSSP is to pick a direction for each edge thereby determining an order for the pair of operations. If all the edges have been ordered, and an acyclic network results, the time associated with the implied schedule is computed by finding the longest path, based on the node weights, in the directed network. If the resulting network is cyclic, the implied schedule is infeasible.

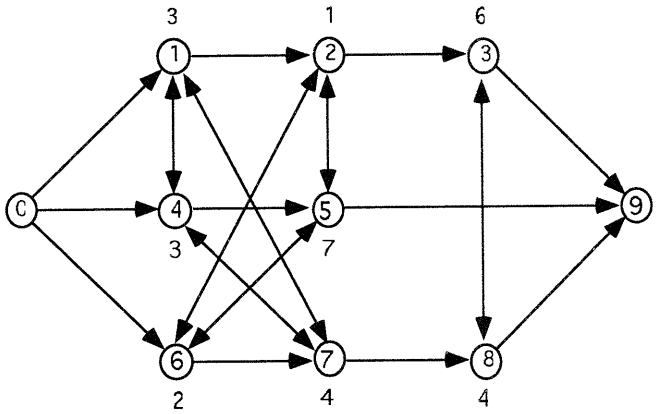
Example Figure 1-1 taken from Lageweg, Lenstra & Rinnooy Kan (1977) depicts a 3–job, 8–operation, disjunctive graph. The jobs are: (1) 1, 2, 3; (2) 4, 5; (3) 6, 7, 8. The eight operations correspond to the nodes in the network. The directed

arcs (the ones with one head) are called conjunctive arcs and reflect the sequence in which operations of a job are to be performed. The machine assignments are: machine (1) 1, 4, 7; machine (2) 2, 5, 6; machine (3) 3, 8. The undirected arcs (the ones with two heads) are called disjunctive arcs and reflect the fact that both operations must be performed on the same machine. The number next to the nodes are the processing times of the operations (Figure 1-1 (a)). Figure 1-1 (b) is an acyclic network representing a feasible (and optimal) schedule. The longest path is 0, 1, 4, 7, 8, 9 with time equal to 14 (Figure 1-1 (c)).

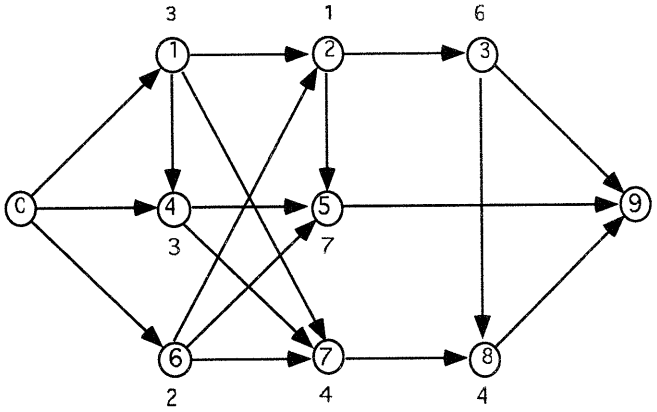
In a typical Branch and Bound scheme, a subproblem would be characterized by a subset of disjunctive arcs that have been resolved with respect to order (direction); the directed network including these arcs and all the conjunctive arcs must be acyclic. The method proceeds to try to fathom this subproblem (that is, find its best completion, or establish that all completions will have higher objective function value than an optimal solution) by optimizing easy to solve relaxations of the residual JSSP, thereby determining lower bounds. If lower bounding fails to fathom a subproblem, two or more new subproblems are created from it by branching on one or more disjunctivearcs. Knowledge about scheduling conflicts and other problem specific information is used in making the branching decisions. These analytic and experimental approaches are reviewed and extended in Lageweg, Lenstra & Rinnooy Kan (1977).

1.3 JSSP and An Assumption-Based Heuristic

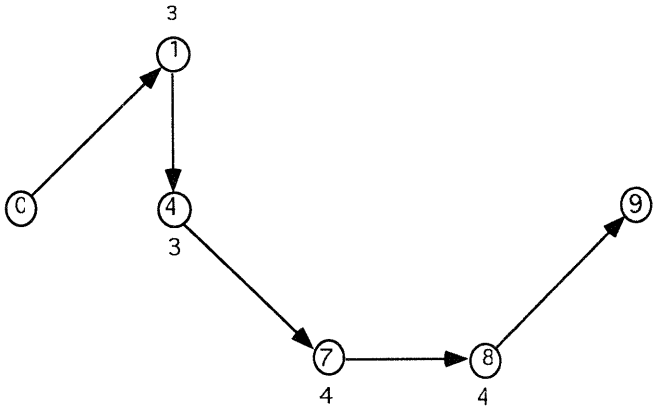
As we mentioned before, heuristic solution procedures for the JSSP also have been widely studied. Most heuristics for the classical JSSP are local search methods. The local search starts with an initial solution. It then searches for neighbors which are solutions generated by a small change from the present solution and substitutes the



(a) The JSSP's disjunctive graph representation



(b) An acyclic network representing a feasible schedule



(c) The optimal schedule of the JSSP = 14 (the longest path)

Figure 1-1: The example

solution. A resurgence of interest in local search has occurred in recent years due to the development of probabilistic local search methods such as simulated annealing, genetic algorithms, and Tabu search. In simulated annealing (Kirkpatrick 1983), substitution in the direction of the worse objective function is allowed in some probability. By controlling this probability, the method escapes from a local optimal solution. Its weak point is that it may take very long to converge. Trying to speed it up can cause the method to be trapped at a local optimum. In short, simulated annealing essentially has the same weak points as iterative improvement.

Here we introduce an assumption-based heuristic approach (Hara 1989) for the JSSP. The idea is to formulate the JSSP as a set of assumptions and to use minimal support for search control.

A solution is assumed to be a set of assumptions. For example, in the JSSP, a pair of jobs processed successively on a machine are an assumption for the problem. The approach's purpose is to find a set of assumptions that satisfies the given constraints and optimizes the objective function. Thus solution S means a set of assumptions.

The system performs the local search while calculating and maintaining a minimal support of the variables and the objective function. A minimal support is the minimal set of assumptions that guarantees the value of a variable. It is a subset of the assumption set that organizes a solution.

The approach has three main advantages: first, any change in a mininal support forces the overall solution to change; second, it reduces the number of neighbors in the local search using minimal support, making its search effective; third, the loop check and search pruning are executed effectively using minimal support, so the system does not control the search by the objective function, i.e. substitution in the direction of the worse objective function is allowed, and does not stay at a local optimal solution.

Minimal support is calculated the same as TMS (Truth Maintenance System) and

ATMS (Assumption-based TMS) (Reiter and de Kleer, 1987). A pair of data elements and minimal support will be referred to as [data, minimal support] from here on. Two examples of minimal support calculation are as follows:

Example 1 [x = 1, X], [y = 5, Y], z = x+y $\Rightarrow$ [z = 6, $X \cup Y$].

Example 2 [x $\geq$ 8, X], [y = 6, Y], z = max(x, y) $\Rightarrow$ [z $\geq$ 8, X].

Minimal support for the objective function of solution S is called minimal support of solution S. In the assumption-based approach, generated minimal supports of solutions are stored as MSset and used at the time of the substitution check called an MScheck. This mechanism defends the loop of the search and prunes search space. The algorithm is as follows:

Algorithm

Step1: Let MSset be empty.

Step2: Get an initial solution S and compute its cost $C(S)$ and minimal support MSs of S.

Step3: Add MSs to MSset.

Step4: Modify S locally into S' that does not include MSs, and compute $C(S')$. If $C(S') < C(S)$, set $S = S'$.

Step5: If a new solution was found in Step4, go to Step2.

Step6: If $C(S)$ is good enough, or if the computational effort thus far has exceeded its bound, terminate with S. Otherwise, determine a new solution S^* and return to Step2.

Heuristics need not improve the overall state of the solution, that is, the JSSP

Table 1.1: The JSSP (processing machine number)

Operation	1	2	3
Job1	1	2	3
Job2	3	1	2
Job3	1	3	2
Job4	2	3	1

Table 1.2: The JSSP (execution time)

Operation	1	2	3
Job1	5	8	2
Job2	7	3	9
Job3	1	7	10
Job4	4	11	7

objective function. It is desirable to guarantee local improvement in minimal support, however.

The following is an example of using the heuristic algorithm to optimize a classical JSSP. It consists of 4 jobs and 3 machines. Each job consists of 3 operations and each operation is processed on a different preassigned machine. O_{jks} denotes the kth operation of job j that is processed on machine s (note the order of the subscripts).

The purpose of the problem is to define the order of operations on each machine. Table 1.1 and Table 1.2 are from Suzuki (1982) (See Figure 1-2). Table 1.3 is the corresponding job sequence indexes for MIP formulations. To apply the heuristic algorithm requires that the assumption be defined. A pair of jobs processed succes-

Table 1.3: The JSSP (job sequence index for MIP formulations)

Operation	1	2	3
Job1	1	2	3
Job2	4	5	6
Job3	7	8	9
Job4	10	11	12

Table 1.4: The JSSP (index for the assumption-based job sequences)

Job Sequence	1	2	3	4
Machine1	1	7	5	12
Machine2	10	2	6	9
Machine3	4	3	11	8

sively on a machine are an assumption for the scheduling problem. $O' \prec O$ denotes the assumption that operation O' is processed just before operation O. The schedule (Figure 1-2 (a)) consists of the following set of assumptions:

Machine1: $O_{111} \prec O_{311}$, $O_{311} \prec O_{221}$, $O_{221} \prec O_{431}$.

Machine2: $O_{412} \prec O_{122}$, $O_{122} \prec O_{232}$, $O_{232} \prec O_{332}$.

Machine3: $O_{213} \prec O_{133}$, $O_{133} \prec O_{423}$, $O_{423} \prec O_{323}$.

As we mentioned before, we use the minimization of the time to complete all jobs as an objective function, which can be defined as follows when e_{jks} and y_{jks} denote the execution and completion times of operation O_{jks}:

$$v = \min F \tag{1.7}$$

$$F = \max_{1 \le j \le n} y_{jms} \tag{1.8}$$

The start time of an operation of a job is the later time between the completion time of the preceding operation of the job and the completion time of the preceding operation processed on the same machine:

$$y_{jks} = \max(y_{j(k-1)s'}, y'_{jks}) + e_{jks} \tag{1.9}$$

In the above equation, y'_{jks} denotes the completion time of operation O' such that $O' \prec O_{jks}$. If operation O' is not found, i.e., O is the first operation on machine s, $y'_{jks} = 0$. Similarly, $y_{j0s} = 0$.

Minimal support of the assumption-based solution is calculated in the paper (Hara, 1989) based on this formulation, as follows:

$$\begin{cases} [y_{111} \geq 5, \{\ \}], \\ [y'_{122} \geq 4, \{\ O_{412} \prec O_{122}\ \}], \\ y_{122} = max(y_{111}, y'_{122}) + e_{122} \Rightarrow\ [y_{122} \geq 13, \{\ \}]. \end{cases}$$

$$\begin{cases} [y_{122} \geq 13, \{\ \}], \\ [y'_{133} \geq 7, \{\ O_{213} \prec O_{133}\ \}], \\ y_{133} = max(y_{122}, y'_{133}) + e_{133} \Rightarrow\ [y_{133} \geq 15, \{\ \}]. \end{cases}$$

$$\begin{cases} [y_{412} \geq 4, \{\ \}], \\ [y'_{423} \geq 15, \{\ O_{133} \prec O_{423}\ \}], \\ y_{423} = max(y_{412}, y'_{423}) + e_{423} \Rightarrow\ [y_{423} \geq 26, \{\ O_{133} \prec O_{423}\ \}]. \end{cases}$$

$$\begin{cases} [y_{311} \geq 6, \{\ O_{111} \prec O_{311}\ \}], \\ [y'_{323} \geq 26, \{\ O_{133} \prec O_{423}, O_{423} \prec O_{323}\ \}], \\ y_{323} = max(y_{311}, y'_{323}) + e_{323} \Rightarrow\ [y_{323} \geq 33, \{\ O_{133} \prec O_{423}, O_{423} \prec O_{323}\ \}]. \end{cases}$$

$$\begin{cases} [y_{323} \geq 33, \{\ O_{133} \prec O_{423}, O_{423} \prec O_{323}\ \}], \\ [y'_{332} \geq 22, \{\ O_{122} \prec O_{232}, O_{232} \prec O_{332}\ \}], \\ y_{332} = max(y_{323}, y'_{332}) + e_{332} \Rightarrow\ [y_{332} \geq 43, \{\ O_{133} \prec O_{423}, O_{423} \prec O_{323}\ \}]. \end{cases}$$

$$\begin{cases} [y_{133} \geq 15, \{\ \}], \\ [y_{232} \geq 22, \{\ O_{122} \prec O_{232}\ \}], \\ [y_{332} \geq 43, \{\ O_{133} \prec O_{423}, O_{423} \prec O_{323}\ \}], \\ [y_{431} \geq 33, \{\ O_{133} \prec O_{423}\ \}], \\ F = max(y_{133}, y_{232}, y_{332}, y_{431}) \Rightarrow\ [F \geq 43, \{\ O_{133} \prec O_{423}, O_{423} \prec O_{323}\ \}]. \end{cases}$$

Consequently, the minimal support of the solution is $\{\ O_{133} \prec O_{423}, O_{423} \prec O_{323}\ \}$. This is, of course, a subset of the assumption set that organizes the solution. Minimal support guarantees that the solution cannot be improved without modifying at least one assumption in it.

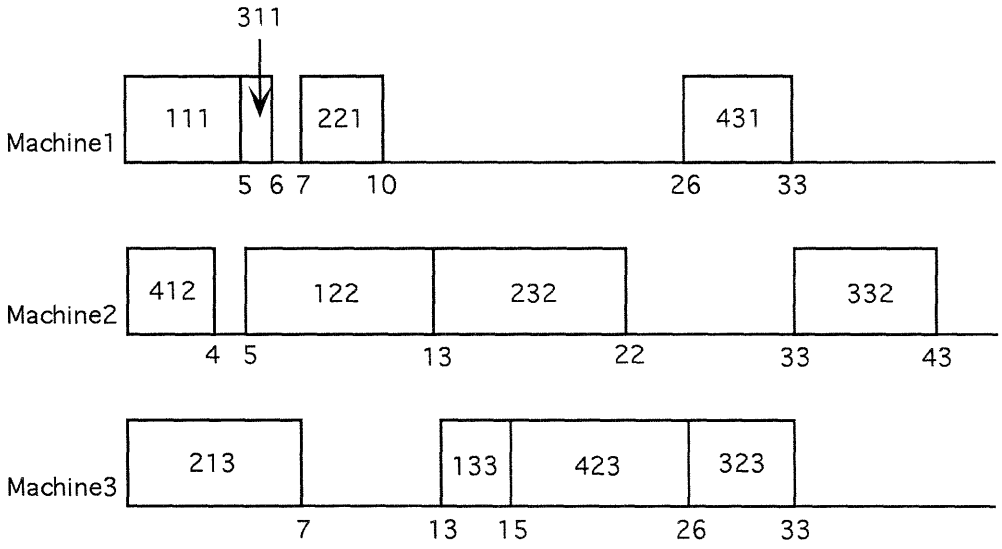
When you try to use the local search method for the JSSP, a simple definition of neighbor is the one that obtains by selecting one operation and moving it into a different position. For a classical $n-$job $m-$machine JSSP, if we use this definition, the number of operations to be selected will be mn and the number of positions to move into will be $n-1$. So the number of neighbors is $mn(n-1)$. The number of the neighbors is large. It is, however, guaranteed that a modification that does not modify any assumptions in minimal support cannot improve the objective function. So, the wasteful search for neighbors can be reduced using information from minimal support.

As described above, the algorithm selects an assumption out of a minimal support and makes modifications. For example, in Figure 1-2 (a), the algorithm may try to change the sequence of O_{423} and O_{323}. Assumption $O_{423} \prec O_{323}$ is taken away and the minimal support is changed. The new solution is shown in Figure 1-2 (b). Minimal support of the solution is $\{ O_{133} \prec O_{323}, O_{323} \prec O_{423} \}$. In modification, the objective function F is improved (the value of F changes from 43 to 40). Such improvement is not, however, guaranteed in the algorithm. The experiments by using the assumption-based heuristic approach have demonstrated that the algorithm (1) operates efficiently even with large problems, (2) finds a good solution at the beginning of the search, and (3) finds a better solution when given a better initial solution (Hara, 1989).

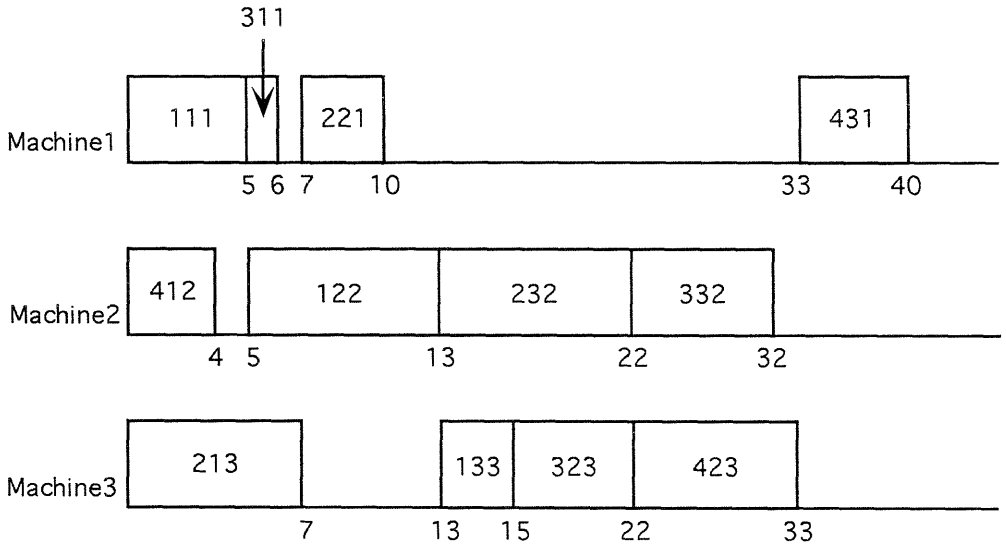
The assumption-based heutistic algorithm can also be interpreted by the disjunctive graph, which transforms JSSPs to longest route problems to be solved by the combination of Branch and Bound schemes and heuristics. The set of assumptions of the job sequences can be expressed in Table 1.4 by the indexes from Table 1.3.

In Section 1.2 we define:

$$x_{jg} = \begin{cases} 1 & \text{if operation } j \text{ performed before operation } g \\ 0 & \text{if operation } g \text{ performed before operation } j \end{cases} \quad \text{for } j,\ g \in J_k$$



(a) Hara's example of schedule



(b) Hara's example of schedule after modification

Figure 1-2: The JSSP schedules

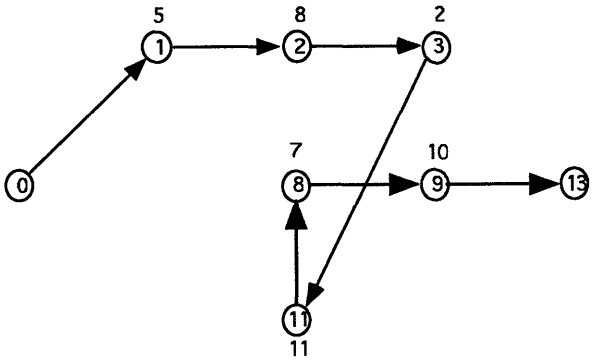
Table 1.5: The JSSP (job sequences on Machines)

Machine1 Variable	x_{17}	x_{15}	x_{112}	x_{57}	x_{712}	x_{512}
Value	1	1	1	0	1	1
Machine2 Variable	x_{210}	x_{610}	x_{910}	x_{26}	x_{29}	x_{69}
Value	0	0	0	1	1	1
Machine3 Variable	x_{34}	x_{411}	x_{48}	x_{39}	x_{38}	x_{811}
Value	0	1	1	1	1	0

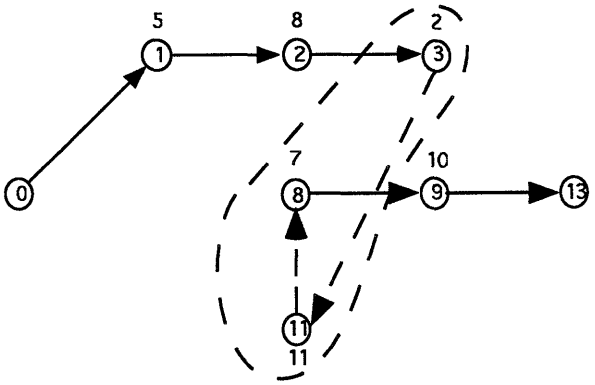
The schedule (Figure 1-2 (a)) consists of the set of assumptions showed in Table 1.5. The assumption-based solution is one of the critical paths of the corresponding longest route problem of the JSSP, which is shown in Figure 1-3 (a). The critical path or the feasible solution is 43, as the same result computed by TMS and ATMS before. The edges (3, 11) and (11, 8) are the minimal support of the assumption-based solution, which can also be found by the corresponding LP relaxation (Figure 1-3(b)). Finally, the heuristic tries to change the sequence of $x_{811} = 0$, which is $O_{423} \prec O_{323}$. Assumption $x_{811} = 0$ is taken away and the minimal support is changed to $x_{811} = 1$, which is $O_{323} \prec O_{423}$. The new solution is shown in Figure 1-3 (c), which is 40 as before.

1.4 Combining MIP and Heuristics

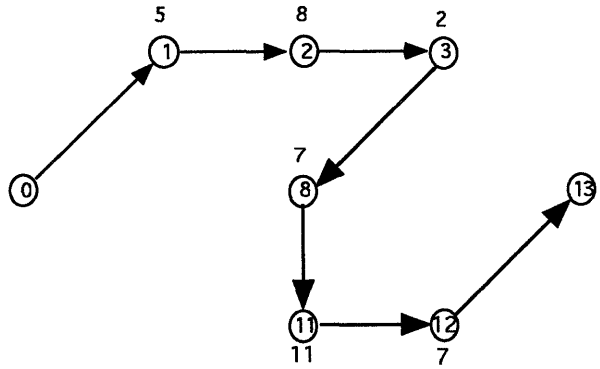
As we mentioned before, the complexity of job-shop scheduling leads to large and difficult combinatorial optimization models. Usually, analytical (and usually computationally intense) methods are used to find optimal solutions and heuristic (and usually computationally simple) methods are used to find good solutions quickly. Often, by using heuristics, it is possible to quickly generate solutions that are feasible and good, but not guaranteed to be optimal. Thus, the practical use of models and associated analytical and heuristic methods should be to identify demonstrably good schedules, rather than to persist in trying to find an optimal solution. In Chap-



(a) The assumption-based solution is represented by the critical path, which is 43.



(b) The minimal support is represented by the edges (3, 11) and (11, 8), which are X_{311}=1 and X_{811}=0.



(c) The heuristic tries to change the minimal support and gets a new critical path, which is 40. The minimal support is changed to X_{311}=1 and X_{811}=1.

Figure 1-3: The disjunctive graph interpretation

ter 2, we introduce how to use the CPLEX Mixed Integer Optimizer, which is an advanced, high-performance computer software package, to solve the JSSP by combining MIP and heuristics (Stopping Branch and Bound short of an optimal solution can be viewed as a heuristic). In Chapter 3, we are going to explore to solve the JSSP by combining Bender's Decomposition and the assumption-based heuristic algorithm.

Chapter 2

Solving the JSSP by MIP

2.1 CPLEX and CPLEX Control Parameters for the Branch and Bound

In this section we use CPLEX to solve MIP formulations of the JSSP. CPLEX is one of the fastest mathematical programming computer software packages. It is an advanced computer software package for LPs and MIPs. It has its own Linear Optimizer, Mixed Integer Solver, Callable Routine Library, and sophisticated preprocessing. When a problem is an LP, a simplex method is used; when a problem contains integer variables (MIP or IP), the Branch and Bound method is used (Using the CPLEX^{TM} Callable Library, 1993).

To find a solution for an MIP or a JSSP, a series of LP subproblems are solved by the CPLEX Branch and Bound method. Each subproblem is a node of the Branch and Bound tree. The root node of the tree is the LP relaxation. According to Using the CPLEX^{TM} Callable Library (1993), the path CPLEX takes through the Branch and Bound tree is controlled by several user setting parameters. For example, at each node, CPLEX can either search deeper into the tree or backtrack. The setting of the

BACKTRACK parameter controls this option. If CPLEX decides to backtrack, there are usually a large number of available, unexplored nodes to be chosen. The NODESELECT parameter setting controls this selection. Williams (1990) summarizes the Branch and Bound method which is featured within the CPLEX Mixed Integer Solver. A more advanced treatment of this method can be found in Nemhauser and Wolsey (1988).

We can set different algorithmic parameters properly to successfully reduce solution times and solve JSSPs. However, determining what parameters to adjust is often difficult to predict. Some experimentation is usually required. For a JSSP, we have found that the values of the two setting parameters, BACKTRACK and NODESELECT, are very important to reduce solution time.

The BACKTRACK parameter determines how frequently backtracking is done during the MIP solving process. The values of BACKTRACK can be any positive number. Low values increase the amount of backtracking, making the search process more of a pure breadth-first search. Higher values (greater than 1.0) decrease backtracking, making the search more of a pure depth-first search. The CPLEX default BACKTRACK value is 0.85. However, we find that decreased backtracking (higher values) often reduces the solution time for the JSSP. To sum up:

$$\text{BACKTRACK=} \begin{cases} \text{default: } 0.85 \\ \text{possible values: any positive number} \end{cases}$$

The NODESELECT parameter determines how to select the next node to process branching when backtracking. According to Using the CPLEX^{TM} Callable Library (1993), NODESELECT has three possible strategies: the depth-first search strategy selects the most recently created node; the best bound strategy selects the node with the best objective function for the associated LP relaxation; and the best esti-

mate strategy selects the node with the best estimate of the integer objective value that would be obtained from a node once all integer infeasibilities are removed. The CPLEX default NODESELECT value is 1. However, we find that the best estimate strategy (the NODESELECT value is 2) reduces the solution time for the JSSP. To sum up:

$$\text{NODESELECT}= \begin{cases} \text{default: } 1 \\ \text{possible values: } \begin{cases} 0 = \text{depth-first search} \\ 1 = \text{best-bound search} \\ 2 = \text{best-estimate search} \end{cases} \end{cases}$$

2.2 Some Experiments Applying CPLEX to JSSP

We applied CPLEX to several JSSP test problems. For this purpose, we implemented a matrix generator to create the MIP formulation (1.1)−(1.6). As we mentioned before, many different CPLEX algorithmic parameters can be employed to successfully reduce solution times and solve the JSSP. However, determining what parameters to adjust is often difficult to predict. Some experimentation is usually required. For a classical JSSP, from our experiments we have found that the values of the two setting parameters, BACKTRACK and NODESELECT, are very important to reduce solution time.

We compare the CPLEX default MIP settings with our improved settings. We apply both settings to a 3−machine 4−job problem, a 5−machine 5−job problem, a 6−machine 6−job problem, a 8−machine 8−job problem, and a 10−machine 10−job problem. The 10−machine 10−job problem is from Muth and Thompson (1963); the others are randomly generated. The results are shown in Table 2.1 and Table 2.2.

For the default settings, The BACKTRACK and NODESELECT are as follows:

Table 2.1: CPLEX MIP Default Settings for JSSPs

JSSP Problem (Machines ×Jobs)	Solution Time (Seconds on Sun SPARCstation 10 Model 41)	Nodes Solved	Best Objective Function Value	Number of Feasible Solutions	Node of First Feasible Solution	Node of Best Feasible Solution
3 ×4	0.45	71	32 (Optimal)	4	14	71
5 ×5	5.72	579	55 (Optimal)	3	7	579
6 ×6	208.28	12429	68 (Optimal)	4	8	9551
8 ×8	1096.17	20000	332(Optimal=72)	1	17	17
10 ×10	3699.55	20000	3194(Optimal=930)	1	20	20

Table 2.2: Our CPLEX MIP Improved Settings for JSSPs

JSSP Problem (Machines ×jobs)	Solution Time (Seconds on Sun SPARCstation 10 Model 41)	Nodes Solved	Best Objective Function Value	Number of Feasible Solutions	Node of First Feasible Solution	Node of Best Feasible Solution
3 ×4	0.43	67	32 (Optimal)	4	14	67
5 ×5	3.93	441	55 (Optimal)	7	7	97
6 ×6	191.90	13373	68 (Optimal)	21	8	1538
8 ×8	753.93	20000	88 (Optimal=72)	23	17	1558
10 ×10	1744.25	20000	1113(Optimal=930)	46	20	4733

$$\text{CPLEX MIP default settings} = \begin{cases} \text{BACKTRACK} = 0.85 \\ \text{NODESELECT} = 1 \end{cases}$$

For our improved settings, we let BACKTRACK and NODESELECT be as follows:

$$\text{Our CPLEX MIP improved settings} = \begin{cases} \text{BACKTRACK} = 1.50 \\ \text{NODESELECT} = 2 \end{cases}$$

In terms of solution times, best objective function values, and number of feasible solutions, the efficiency and the quality of the solutions of our improved settings are much better than those of the default settings. The performance of CPLEX on the 10−machine 10−job test problem was still not very good.

Chapter 3

Combining Benders' Decomposition and Heuristic

3.1 Benders' Decomposition for MIP and JSSP

Benders' Decomposition has often been proposed and sometimes used to decompose the MIP:

$$\begin{aligned} v = \min\ & \mathbf{cx}+\mathbf{hy} \\ & \text{subject to} \\ & \mathbf{Ax}+\mathbf{Qy} \leq \mathbf{b} \\ x_j = 0 \text{ or } 1 \quad & \mathbf{y} \geq \mathbf{0} \end{aligned} \tag{3.1}$$

The decomposition of (3.1) consists of an integer programming master problem involving the $\mathbf{x}$ variables, and a linear programming subproblem involving the $\mathbf{y}$ variables. The approach is resource directive because for $\tilde{\mathbf{x}}$ fixed at $0-1$ values, (3.1) reduces

to the linear programming subproblem:

$$\begin{aligned} v(\tilde{\mathbf{x}}) = \min\ & \mathbf{hy} \\ & \text{subject to} \\ \mathbf{Qy} \leq\ & \mathbf{b} - \mathbf{A}\tilde{\mathbf{x}} \\ & \mathbf{y} \geq \mathbf{0} \end{aligned} \tag{3.2}$$

and $\mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}$ is the resource available for this minimization.

We consider Benders' Decomposition for MIP and the dual to (3.2)

$$\begin{aligned} v(\tilde{\mathbf{x}}) = \max\ & \mathbf{u}(\mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}) \\ & \text{subject to} \\ & \mathbf{uQ} \leq \mathbf{h} \\ & \mathbf{u} \leq \mathbf{0} \end{aligned} \tag{3.3}$$

The function $v(\mathbf{x})$ is a nondifferentiable convex function and the MIP (3.1) can be written as the nonlinear integer problem:

$$\begin{aligned} v = \min\ & \mathbf{cx} + v(\mathbf{x}) \\ & \text{subject to} \\ & x_j = 0 \text{ or } 1 \end{aligned}$$

The piecewise linear nature of v permits us to convert this problem into a linear integer programming problem.

Let $\mathbf{u}^t$ for $t = 1, ..., T$, denote the extreme points of the dual feasible region in (3.3)

and let $\mathbf{u}^k$ for $k = 1, ..., K$, denote the extreme rays. At any arbitrary iteration of the Benders' algorithm, we use subsets of the dual extreme points and rays to construct the integer programming master problem:

$$\begin{aligned} v' &= \min v \\ &\text{subject to} \\ v \geq \mathbf{cx} + \mathbf{u}^t(\mathbf{b} - \mathbf{Ax}) \quad & t = 1, ..., T' \qquad (3.4) \\ \mathbf{u}^k(\mathbf{b} - \mathbf{Ax}) \leq 0 \quad & k = 1, ..., K' \qquad (3.5) \\ x_j &= 0 \text{ or } 1 \end{aligned}$$

This problem is essentially a pure IP problem that can be solved. It suffices to assume that some algorithm can be used to produce an optimal solution $\tilde{\mathbf{x}}$. It can be shown that v' is a lower bound on the minimal MIP cost v. The solution $\tilde{\mathbf{x}}$ is used in the pair of linear programming problems (3.2) and (3.3) which are optimized by the simplex method. If (3.2) is infeasible, then a new dual extreme ray is discovered and a constraint is added to the set (3.5). If (3.2) is feasible with right-hand-side $\mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}$, then it has an optimal solution $\tilde{\mathbf{y}}$ and $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ is a feasible solution to the MIP (3.1). Let $\tilde{\mathbf{u}}$ denote the optimal solution to (3.3) found by the simplex method. The solution $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ is optimal in (3.1) if $v' \geq \mathbf{c}\tilde{\mathbf{x}} + \mathbf{h}\tilde{\mathbf{y}} = \mathbf{c}\tilde{\mathbf{x}} + \tilde{\mathbf{u}}(\mathbf{b} - \mathbf{A}\tilde{\mathbf{x}})$. If this optimality test fails, then the constraint:

$$v \geq \mathbf{cx} + \tilde{\mathbf{u}}(\mathbf{b} - \mathbf{Ax})$$

is added to the set (3.4).

Benders' algorithm for the MIP converges in a finite number of iterations to an

optimal solution because each time the integer programming master problem (3.4) and (3.5) is solved, there is a new constraint (or, cut) in (3.4) or (3.5), and there are only a finite number of such constraints possible. The algorithm has the desirable feature of producing a feasible solution to (3.1) at each iteration that (3.2) is feasible, and the lower bounds v' on the cost of an optimal solution in (3.1). Moreover, the lower bounds are monotonically increasing with iterative solutions of the master problem (Shapiro, 1979).

To apply the Benders' algorithm for the MIP discussed above to the JSSP formulated as an MIP in Section 1.2, we let $\mathbf{c} = \mathbf{0}$, $\mathbf{h} = (\mathbf{0},1)$, and $\mathbf{y} = (\mathbf{t}, F)^T$ in (3.1), where $\mathbf{t}$ is the vector of start times for operations and F is equal to the maximum of the completion times of all jobs. Now, the problem is how to find a good feasible solution $\tilde{\mathbf{x}}$ to the JSSP version of (3.1) and to solve the JSSP. In the next section, we are going to explore how to combine Benders' algorithm with the assumption-based heuristic algorithm discussed in Section 1.3 to solve the MIP formulations for the JSSP.

3.2 A Combined Algorithm for JSSP

As we mentioned before, Benders' algorithm for the JSSP converges in a finite number of iterations to an optimal solution. Moreover, the lower bounds are monotonically increasing with iterative solutions of the master problem. And the assumption-based heuristic algorithm is possible to quickly generate solutions for the JSSP that are feasible and good, but not guaranteed to be optimal. Those features lead to a basic theme of this thesis. An upper bound is determined by any feasible solution $\tilde{\mathbf{x}}$ ($0-1$ variables) to the original JSSP (or, the JSSP's subproblem of the Benders' algorithm), which can be generated by the assumption-based heuristic algorithm. The integer solution corresponding to the best upper bound encountered in the search is the

integer optimum. A lower bound is determined by the optimal solution to the Benders' master problem (3.4) and (3.5). Moreover, the optimal solution to the Benders' master problem can be a set of new assumptions (or new seed) to the JSSP. Figure 3-1 and Figure 3-2 show the basic idea and the flow chart of the combined algorithm.

We used CPLEX to implement the combined algorithm. Instead of using TMS and ATMS to calculate minimal support (see Section 1.3), we can find minimal support by solving the subproblem, which is an LP, after getting an initial solution that is assumed to be a set of assumptions to the JSSP. The LP subproblem's solution is either optimal or infeasible; and its LP dual is either optimal or unbounded (cannot be infeasible). So, feasible dual prices of the LP dual always exist. From the tightest dual prices' constraints of the LP subproblem or its LP dual (if the LP subproblem is infeasible), the minimal support of the assumptions can always be found. By the assumption-based heuristic algorithm, the upper bound of the JSSP is guaranteed to be changed if the minimal support changes. At the same time, the corresponding Benders' master problem is constructed by storing the cuts from the LP subproblem. A control parameter N is set up to solve the master problem (MIP) regularly (how often to solve the master problem depends on the different JSSPs). It serves three important purposes:

(1) to avoid getting the same minimal support so that the algorithm will not become trapped at the same upper bound;

(2) to set up (or to reseed) a new set of assumptions for the JSSP;

(3) to get a lower bound for the JSSP.

The combined algorithm is as follow:

The Combined Algorithm

Step1: Let MSset be empty.

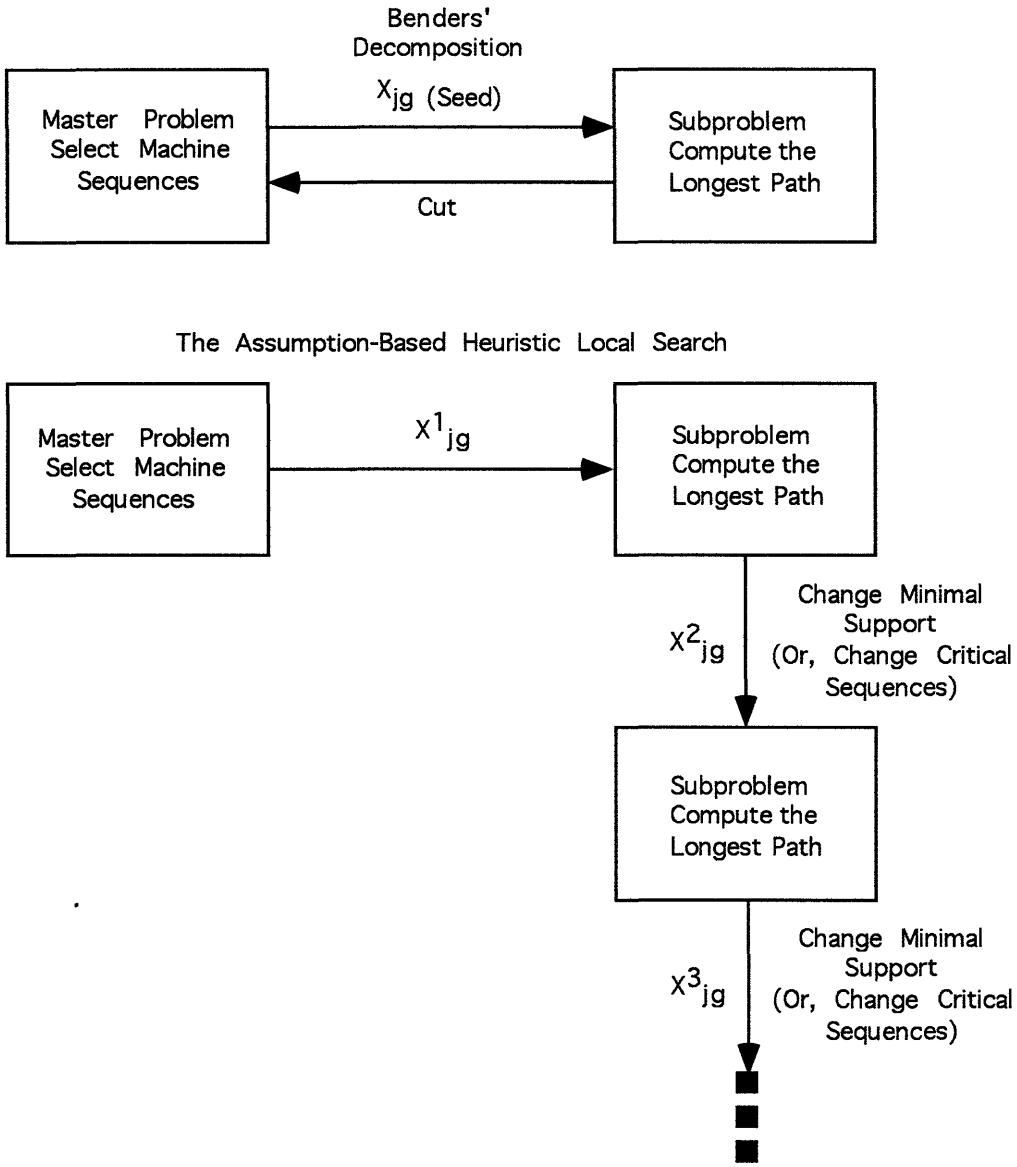


Figure 3-1: Comparison of Benders' Decomposition and local search

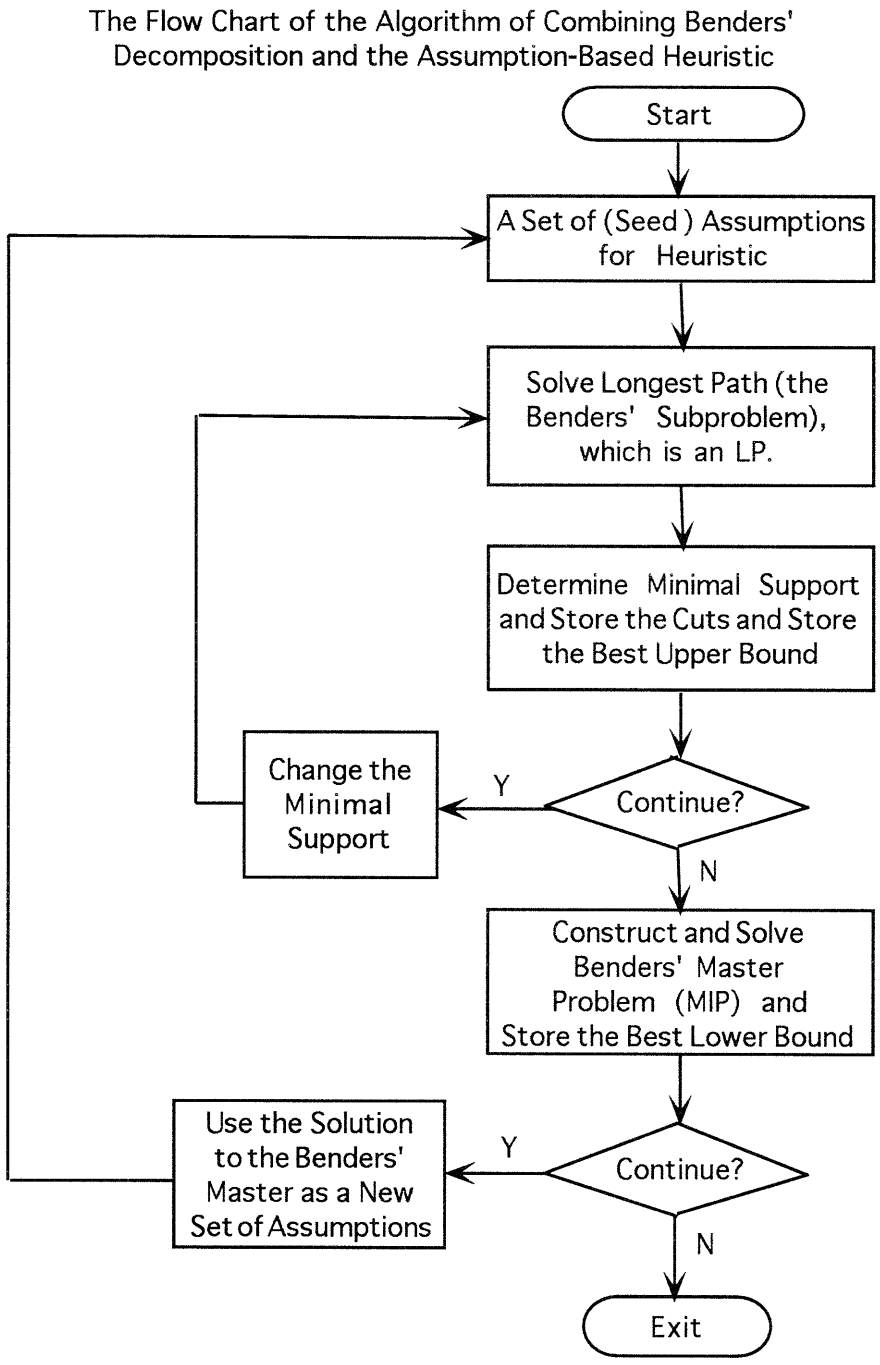


Figure 3-2: The flow chart of the combined algorithm

Step2: Get an initial solution S and use CPLEX to compute its cost $F = C(S)$ and minimal support MSs of S.

Step3: Add MSs to MSset.

Step4: Modify S locally into S' that does not include MSs, and compute $F = C(S')$. If $F = C(S') < C(S)$, save the best upper bound $F = (S')$ so far. At the same time, set $S = S'$ and construct (store the cuts for) Benders' master problem.

Step5: If a new solution was found in Step4, go to Step2.

Step6: Regularly solve (the frequency is controled by a parameter N) Benders' master problem. Let $v = C(\mathbf{x})$ be the cost function. Every time the master problem is solved, save the best lower bound, which is $v = \max\{C(\mathbf{x})\}$. The corresponding solutions $\tilde{\mathbf{x}}$ construct a new initial solution, then go to Step2.

Step7: If $F = C(S)$ or $v = C(\mathbf{x})$ is good enough, or if the computational effort thus far has been too great, terminate with S. Otherwise, determine a new solution S^* and return to Step2.

At Step4, the assumption to be modified should be selected out of MSs. This reduces the number of neighbors to search for. If the result of modification is infeasible for CPLEX to compute F, the algorithm will compute the dual problem (it is unbounded) and get a new minimal support, then go to Step4 again.

At Step6, the mechanism defends the loop that makes the same solution as before and prunes the search space because a solution including the same minimal support as generated before is no longer accepted. The changes of the upper bounds from the assumption-based heuristic algorithm and the lower bounds from Benders' master problem with the iterations of the combined algorithm are shown in Figure 3-3.

Finally, at Step7, no matter how good the solution is or how great the computational effort has been made, a stopping criterion L has been set, which is the number

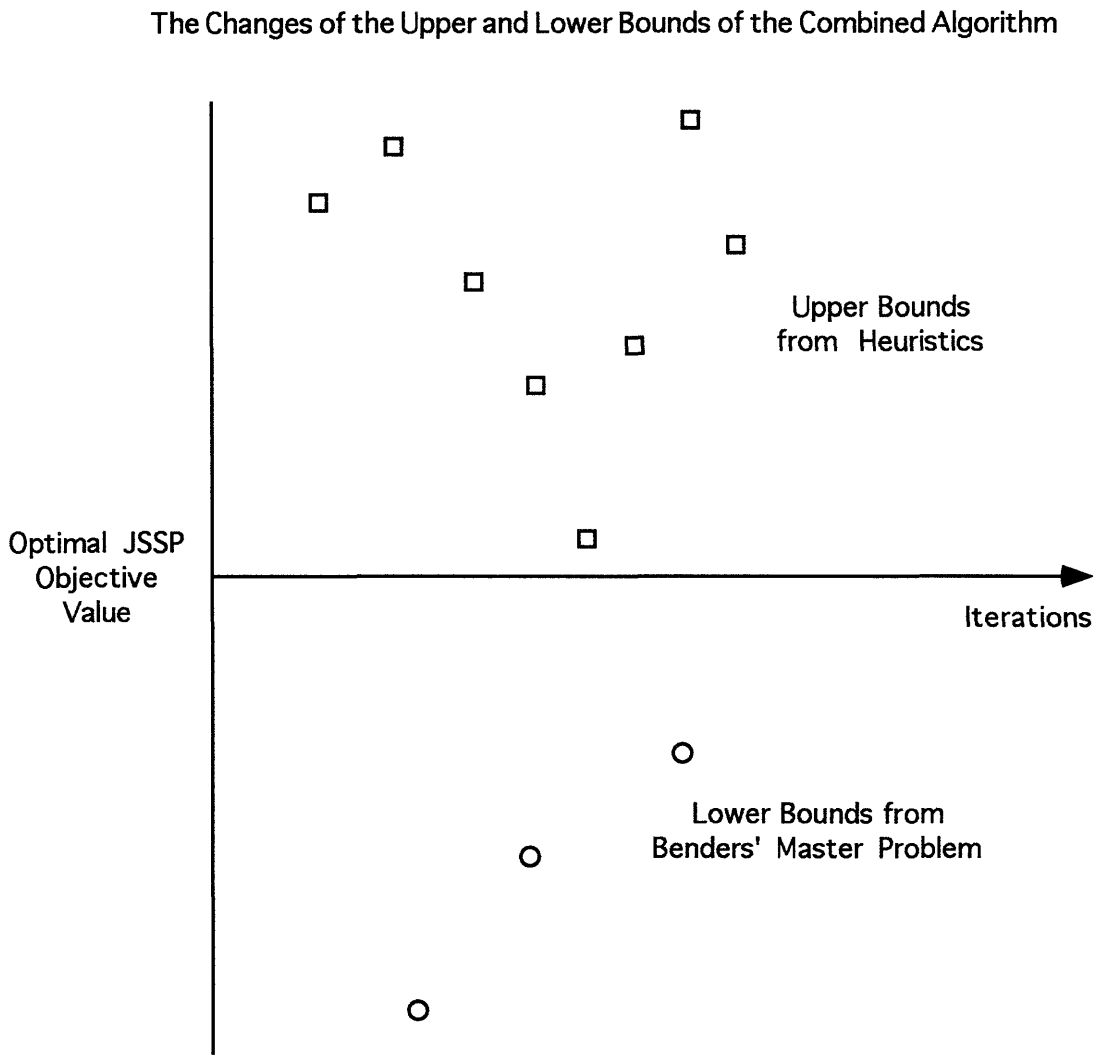


Figure 3-3: The upper and lower bounds

of iterations of solving the subproblem generated by the combined algorithm.

3.3 Experiments and Results

In this section the experiments and results of the combined algorithm described in Section 3.2 are presented. Two sets of experiments were run by the CPLEX^{TM} Callable Library. All experiments were performed on the same JSSPs in the Section 2.2. First, experiments were conducted to determine appropriate value of the combined algorithm control parameters, which are BACKTRACK, NODESELECT, N

Table 3.1: The Results of the Combined Algorithm for JSSPs

JSSP Problem (Machines ×Jobs)	Optimum JSSP Objective Value	The Best Upper Bound (BUB)	Number of Iterations to Get BUB	The Best Lower Bound (BLB)	Number of Iterations to Get BLB
3 ×4	32	32	31	32	35
5 ×5	55	58	1	-10	6
6 ×6	68	80	3	-27	5
8 ×8	72	93	1	68	5
10 ×10	930	1319	1	597	3

(a control parameter for the Benders' master parameter), and L (a stopping control parameter for the algorithm). Next, the performance of the combined algorithm is tested.

Since the Benders' master problem, which is an MIP, is solved repeatedly in our combined algorithm, determining appropriate values for those contol parameters becomes very important to reduce solution time and to get a good solution. The results indicate the following settings are good:

$$\text{The combined algorithm parameter settings } = \begin{cases} \text{BACKTRACK} = 1.20 \\ \text{NODESELECT} = 2 \\ \text{N} = 25 \\ \text{L} = 200 \end{cases}$$

With those settings, we applied the combined algorithm to the same JSSPs in the Section 2.2. The results are abtained in Table 3.1.

3.4 Considerations and Conclusion

The combined algorithm for the JSSP works well on the small size problems, but does not always work well for the large size problems. It mainly depends on whether both the size of the Benders' master problem (MIP) and the size of minimal support can be small enough compared to the number of assumptions that organize a solution. It thus

must be considered, in general, how Benders' master problem can be solved efficiently, how often Benders' master problem should be solved, and how assumptions can be defined to reduce the size of minimal support (thereby reducing the number of cuts stored in Benders' master problem). Of course, the increased size of the Benders' master problem and the minimal support will mainly reduce overall the combined algorithm's efficiency. What takes the most time is to solve Benders' master problem. It takes time because the master problem is simply an MIP. It may take less time if we can adjust some CPLEX Mixed Integer Solver Parameters, or if we can solve it less often. Finally, whether our combined algorithm can prevent cycling between the subproblem and the master problem is unknown to us.

We can see that the lower bounds obtained by the combined algorithm are not very good. The reason is that Benders' algorithm for the MIP or JSSP converges very slowly. The results of the upper bounds abtained by the combined algorithm are encouraging. As we pointed out earlier, the assumption-based heuristic finds a better solution (or a better upper bound) when given a better initial solution. We used the "increasing-time-sequence" rule to modify the initial sets of the assumptions of the same 5×5 and 6×6 JSSPs. For the 5×5 JSSP, the best upper bound is improved from 58 to 56. For the 6×6 JSSP, the best upper bound is improved from 80 to 73.

Chapter 4

Future Research

We would like to explore methods to improve our algorithm, which combines MIP and the assumption-based heuristic for the JSSP. Our experiments and the latest breakthroughs in MIP, heuristics, and computer technology suggest the future research in the JSSP from the following three areas: (1) MIP; (2) heuristics; (3) combining MIP and heuristics.

4.1 MIP

The variety of MIP formulations for the JSSP provides a robust environment for computational research. Recently, some MIPs with thousands of $0-1$ variables have been solved to optimality on a workstation or PC. There are many reasons for this vastly improved capability. To name a few: (1) improved LP technology; (2) good formulations; (3) preprocessing; (4) LP-based heuristics. However, the overall idea of using Branch and Bound with LP relaxation has not changed (Nemhauser, 1994).

Because an LP is solved for each node of the Branch and Bound tree, the improved LP technology has had a major impact. For example, CPLEX has one of the fastest LP Solvers. The simplex method is used to solve the subsequent LPs, because they

are formed by adding a constraint to an already optimized LP. One area of future JSSP research will be to develop faster LP methods. For example, interior-point methods might be taken into consideration.

The LP relaxation of an MIP is a good approximation of the convex hull of integer solutions if the MIP formulation is good. Most MIPs (including JSSPs) have many correct formulations and the failure to solve an MIP or a JSSP may not be the fault of an algorithm but the fault of an improper formulation. One area of future JSSP research will be to construct better JSSP formulations. For example, the alternative-objective-function method can be applied to formulate a better JSSP model (Shapiro, 1993).

An original MIP may be analyzed by preprocessing, which attempts to fix or eliminate variables and redundancies, tighten bounds, and tighten constraints by modifying coefficients or reformulating. Preprocessing can substantially reduce problem size and improve bound quality. Preprocessing can make a hard MIP much easier to solve. In fact, some MIPs could not be solved by conventional Branch and Bound (Nemhauser, 1994), but required no Branch and Bound after a comprehensive preprocessor. Thus, one of the future areas of research will be how to preprocess the Benders' master problem to improve the initial formulation by eliminating redundancies. Since, from our experiments, the Benders' master problem has always many redundancies.

The final issue LP-based heuristics will be discussed in the next two sections.

4.2 Heuristics

As we discussed before, heuristics are often possible to quickly generate JSSP solutions that are feasible and good, but not guaranteed to be optimal. In this thesis, we use the assumption-based heuristic algorithm to generate feasible integral solutions, possibly

decreasing the current upper bound. This approach belongs to Primal Heuristics. One of the future areas of research can be, instead of using Benders' Decomposition to get the lower bound and a new set of assumptions, to use Dual Heuristics which can often generate feasible solutions to the LP relaxation dual (hence lower bounds) faster than both the LP solver and the MIP solver, or which can generate dual solution to differant relaxations.

The assumption-based heuristic algorithm introduced in this thesis is a local search method controled by minimal support. The method is very practical because it operates efficiently even with large problems, finds a good solution at the beginning of the search, and finds a better solution when given a better initial solution. Thus, one of the future areas of research may be to try to find a good initial solution to the assumption-based heuristic algorithm. We might use Problem-Specific Heuristics as suggested by Panwalker and Iskander (1977) who list over 100 problem specific heuristic rules.

We think that the key future heuristic research in the JSSP is to incorporate the Problem-Specific Heuristics into the Local-Search Heuristics. The result will be to cluster good solutions close together, thus making it easier to perform local search. Probabilistic search methods such as simulated annealing and genetic algorithms can be applied directly to the search spaces.

4.3 Combining MIP and Heuristics

In this thesis we combined Benders' Decomposition algorithm and the assumption-based heuristic algorithm, and applied the combined algorithm to optimize the JSSP. As we discussed in the last two sections, our future JSSP research may be to try to improve our current combined algorithm by using the latest developments in MIP and heuristics. Actually, the future JSSP research is very rich and widely open.

Bibliography

[1] Adams, J., E. Balas and D. Zawack (1988), "The Shifting Bottleneck Procedure for Job-Shop Scheduling." *Management Science, Vol. 34*, pp. 391-401.

[2] Afentakis, P. and B. Gavish (1986), "Optimal Lot-Sizing Algorithms for Complex Product Structures." *Operations Research, Vol. 34*, pp. 237-249.

[3] Akers, S.B. and J. Friedman (1955), "A Non-Numerical Approach to Production Scheduling Problems." *Operations Research, Vol. 3, No. 4*, pp. 429-442.

[4] Balas, E. (1979), "Disjunctive Programming." *Ann. Discrete Math, Vol. 5*, pp. 3-51.

[5] Bradley, S.P. et al. (1977), *Applied Mathematical Programming.* Addison-Wesley Publishing Company, Reading, MA.

[6] Clemmer, G.L. (1984), *An Artificial Intelligence Approach To Job-Shop Scheduling.* M.S. Thesis, The Sloan School of Management, MIT.

[7] De Kleer, J. (1986), "An Assumption-based TMS." *Artificial Intelligence, Vol. 28*, pp. 127-162.

[8] De Kleer, J. (1986), "Extending the ATMS." *Artificial Intelligence, Vol. 28*, pp. 163-196.

[9] De Kleer, J. (1986), "Problem Solving with the ATMS." *Artificial Intelligence, Vol. 28*, pp. 197-224.

[10] Erschler, J. et al. (1983), "A New Dominance Concept in Scheduling n Jobs on a Single Machine with Ready Times and Due Dates." *Operations Research, Vol. 31, No. 1*, pp. 114-127.

[11] French, S. (1982), *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop.* Horwood, Chichester.

[12] Giffler, B. et al. (1963), "Numerical Experience with the Linear and Monte Carlo Algorithms for Solving Production Scheduling Problems." *Industrial Scheduling, edited by J.F. Muth and G.L. Thompson.* Prentice-Hall, Inc., Englewood Clifts, NJ, pp. 21-38.

[13] Giffler, B. and G.L. Thompson (1960), "Algorithms for Solving Production Scheduling Problems." *Operations Research, Vol. 8*, pp. 487-503.

[14] Graves, S.C. et al., Eds (1993), *Handbooks in OR & MS, Vol. 4.* Elsevier Science Publishers B.V.

[15] Hara, H. et al. (1989), "An Assumption-Based Combinatorial Optimization System." Artificial Intelligence Laboratory, Fujitsu Laboratories Ltd., Kawasaki, Japan.

[16] Kirkpatrick, S. et al. (1983), "Optimization by Simulated Annealing." *Science*, pp. 220-225.

[17] Kumara, S.T. et al., Eds (1989), *Artificial Intelligence: Manufacturing Theory And Practice.* The Institute of Industrial Engineers.

[18] Lageweg, B., J.K. Lenstra and A.H.G. Rinnooy Kan (1977), "Job Shop Scheduling by implicit enumeration." *Management Science, Vol. 24*, pp. 441-450.

[19] Muth, J.F. and G.L. Thompson (1963), *Industrial Scheduling.* Prentice-Hall, Englewood Cliffs, NJ.

[20] Nemhauser, G.L. (1994), "The Age of Optimization: Solving Large-Scale Real-World Problems." *Operations Research, Vol. 42, No. 1*, January-February.

[21] Nemhauser, G.L. and L.A. Wolsey (1988), *Integer and Combinatorial Optimization.* Wiley & Sons, New York.

[22] Panwalkar, S.S. and W. Iskander (1977), "A Survey of Scheduling Rules." *Operations Research, Vol. 23, No. 1*, pp. 45-61.

[23] Reiter, R. and J. de Kleer (1987), "Foundations of Assumption-based Truth Maintenance System." *Proceedings AAAI-87.*

[24] Roy, B. and B. Sussmann (1964), "Les Problems D'ordonnancement Avec Constaintes Disjonctives." *Note DS No. 9 bis, SEMA.*

[25] Schrage, L. (1986), *Linear, Integer and Quadratic Programming with LINDO, 3rd Edition.* Scientific Press, Palo Alto, CA.

[26] Shapiro, J.F. (1993), "Mathematical Programming Models and Methods for Production Planning and Scheduling." *Handbooks in OR and MS, Vol. 4*, pp. 371-443.

[27] Shapiro, J.F. (1979), *Mathematical Programming.* John Wiley & Sons, Inc.

[28] Storer, R.H. et al. (1992), "New Search Space for Sequencing Problems with Application to Job Shop Scheduling." *Management Science, Vol. 38, No. 10*, October.

[29] Suzuki, H. (1982), *Integer Programming and Combinatorial Optimization.* Union of Japanese Scientists and Engineers Press.

[30] Using the CPLEX^{TM} Callable Library (1993), CPLEX Optimization, Inc., Incline Village, NV.

[31] Williams, H.P. (1990), *Model Building in Mathematical Programming.* Wiley & Sons, New York.

[32] Zweben, M. and M.S. Fox (1994), *Intelligent Scheduling.* Morgan Kaufmann Publishers, San Francisco, CA.