Spring 5-31-1997

Scheduling in assembly type job-shops

Nutthapol Asadathorn

New Jersey Institute of Technology

ABSTRACT

SCHEUDLING IN ASSEMBLY TYPE JOB-SHOPS

by
Nutthapol Asadathorn

Assembly type job-shop scheduling is a generalization of the job-shop scheduling problem to include assembly operations. In the assembly type job-shops scheduling problem, there are *n* jobs which are to be processed on *m* workstations and each job has a due date. Each job visits one or more workstations in a predetermined route. The primary difference between this new problem and the classical job-shop problem is that two or more jobs can merge to form a new job at a specified workstation, that is job convergence is permitted. This feature cannot be modeled by existing job-shop techniques. In this dissertation, we develop scheduling procedures for the assembly type job-shop with the objective of minimizing total weighted tardiness. Three types of workstations are modeled: single machine, parallel machine, and batch machine. We label this new scheduling procedure as SB. The SB procedure is heuristic in nature and is derived from the shifting bottleneck concept. SB decomposes the assembly type job-shop scheduling problem into several workstation scheduling sub-problems. Various types of techniques are used in developing the scheduling heuristics for these sub-problems including the greedy method, beam search, critical path analysis, local search, and dynamic programming.

The performance of SB is validated on a set of test problems and compared with priority rules that are normally used in practice. The results show that SB outperforms the priority rules by an average of 19% - 36% for the test problems. SB is extended to solve

scheduling problems with other objectives including minimizing the maximum completion time, minimizing weighted flow time and minimizing maximum weighted lateness. Comparisons with the test problems, indicate that SB outperforms the priority rules for these objectives as well.

The SB procedure and its accompanying logic is programmed into an object oriented scheduling system labeled as LEKIN. The LEKIN program includes a standard library of scheduling rules and hence can be used as a platform for the development of new scheduling heuristics. In industrial applications LEKIN allows schedulers to obtain effective machine schedules rapidly. The results from this research allow us to increase shop utilization, improve customer satisfaction, and lower work-in-process inventory without a major capital investment.

SCHEDULING IN ASSEMBLY TYPE JOB-SHOPS

by
Nutthapol Asadathorn

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

Department of Industrial and Manufacturing Engineering

May 1997

APPROVAL PAGE

SCHEDULING IN ASSEMBLY TYPE JOB-SHOPS

Nutthapol Asadathorn

[illegible] 3/29/97

Dr. Xiuli Chao, Dissertation Advisor — Date
Associate Professor of Industrial and Manufacturing Engineering, NJIT

[illegible] 3/30/97

Dr. Sanchoy Das, Dissertation Co-Advisor — Date
Associate Professor of Industrial and Manufacturing Engineering, NJIT

[illegible] 4/3/97

Dr. Layek Abdel-Malek, Committee Member — Date
Professor of Industrial and Manufacturing Engineering, NJIT

[illegible] 3/31/97

Dr. Carl Wolf, Committee Member — Date
Professor of Industrial and Manufacturing Engineering, NJIT

[illegible] 3/31/97

Dr. Cheickna Sylla, Committee Member — Date
Associate Professor of Industrial Management, NJIT

[illegible] 4/3/97

Dr. Lazar Spasovic, Committee Member — Date
Associate Professor of Transportation, NJIT

BIOGRAPHICAL SKETCH

Author: Nutthapol Asadathorn

Degree: Doctor of Philosophy

Date: May 1997

Undergraduate and Graduate Education:

- Doctor of Philosophy in Industrial Engineering
  New Jersey Institute of Technology, New Jersey, May 1997

- Master of Engineering in Operations Research and Industrial Engineering
  Cornell University, New York, May 1990

- Bachelor of Engineering in Industrial Engineering
  Chulalongkorn University, Thailand, April 1988

Major: Industrial Engineering

Presentations and Publications:

Nutthapol Asadathorn and Xiuli Chao,
"A Decomposition Approximation for Assembly-Disassembly Types of Queuing Networks,"
Submitted to Annals of Operations Research.

Xiuli Chao, Sanchoy Das and Nutthapol Asadathorn,
"A Scheduling Information System on Microsoft Windows,"
INFORM Conference, Atlanta, November 3-6, 1996.

Nutthapol Asadathorn, Xiuli Chao and Sanchoy Das,
"Traveling Salesman Problem with Chain-Type Precedent Constraints,"
International Conference On Management Science and the Economic Development of China, Hong-Kong, July 16-19, 1996.

iv

Layek Abdel-Malek and Nutthapol Asadathorn,
"An Analytical Approach to Process Planning with Rework Option,"
International Journal of Production Economics. 46-47(1996) 551-520.

Carl Wolf and Nutthapol Asadathorn,
"How to Determine The Appropriate Number of Cost Drivers for ABC Product Cost System – An Analytical Method,"
POMS National Conference, Pittsburgh, PA, Octorber 7-10, 1995.

Layek Abdel-Malek and Nutthapol Asadathorn,
"Process and Design Tolerance in Relation to Manufacturing Cost: A Review and Extension,"
The Engineering Economist. Fall 40(1): 73-100.

Sirijan Tongprasert, Thachapol Posayanon and Nutthapol Asadathorn,
Reliability (Thai), 1993.

This dissertation is dedicated to
my beloved wife, Lek

ACKNOWLEDGMENT

I thank my advisors, Professor Xiuli Chao and Professor Sanchoy Das, for their suggestions, guidance and positive criticism throughout this research. Their friendly attitude and subtle discussions during the preparation of the research are very much appreciated. Thanks are extended to Professor Layak Abdel-Malek and Professor Carl Wolf for their supervision and thoughtfulness during my study at New Jersey Institute of Technology. Special thanks to Professor Cheickna Sylla and Professor Lazar Spasovic, my dissertation committee members, for their suggestive comments.

Gratitude is expressed to the Department of Industrial and Manufacturing Engineering, New Jersey Institute of Technology for partial financial support during my stay at the school and Dr. Michael L. Pinedo at Columbia University for his guidance and financial supports.

TABLE OF CONTENTS

Chapter Page

Chapter Page

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

Table Page

CHAPTER 1

INTRODUCTION

Assembly shop may be considered as a variation of the classical job shop. Each assembly job begins as several sub-jobs. These sub-jobs then progress on several paths. At each convergence point (assembly station), two or more sub-jobs (components) merge to form a semi-finished part (see Figure 1-1). At an assembly station, processing operations can start only when all the required components are available. Figure 1-2 shows that operation 3 can start only after operations 1 and 2 have completed. In a mixed model assembly shop, several workstations will process a variety of jobs and/or assembly operations. In such scenario, the sequencing and scheduling policy has a significant impact on shop performance. Sequencing decision at an upstream station will effect all downstream stations. Clearly, developing an effective assembly shop scheduling system requires combining traditional job shop method with flow shop method.

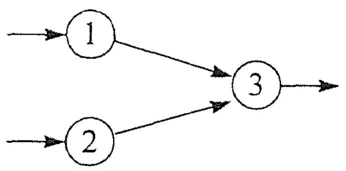


Figure 1-1: Assembly operation

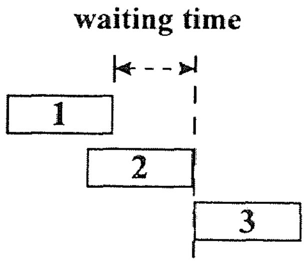


Figure 1-2: Tasks synchronizing

The machines in assembly shop are grouped into workstations. The workstation is a processing stage in the shop. It may consist of single machine or a group of similar machines working in parallel. In this research, we are interested in three types of workstations – single machine, parallel machine and batch machine workstations. A batch machine processes a fixed lot of jobs simultaneously and does not begin processing a new lot unless the previous lot has been completed. For all workstation types, we do not allow preemption of jobs, but do permit job reentrance to the previously visited machines.

When an assembly shop is controlled as a traditional job shop, this may lead to one of the most common problems in production planning and control, the work-in-process inventory. When jobs are considered as multiple independent sub-jobs to prevent assembly structure, work-in-process inventory is unavoidable. This inventory assures the smooth production. Therefore, the work-in-process inventory control problem may be solved by improving the assembly shop control. The storage space and inventory cost can be reduced significantly. The flow will be smooth as long as machines are working as expected.

1.1 Problem Description

A typical assembly shop consists of $j = 1, \ldots, n$ jobs, and there are $k = 1, \ldots, v$ machines in the shop. In such a situation, the scheduler needs to find the processing sequences for all the machines in the shop in order to maximize the customer satisfaction. Here, we determine the customer satisfaction level by measuring the ability to complete the job on time. If a job cannot be completed on time, it is considered as a *delayed job*. Among all customers, some of them may have higher priorities than others. Their orders are important and should not be delayed or, if not avoidable, be delayed at the minimal. Job weights are assigned according to these priorities. In this dissertation, we measure customer satisfaction by weighted tardiness, *WT*. Where $WT = \sum_{j=1}^{n} w^{(j)} T^{(j)}$ and $T^{(j)} = \max(0, d^{(j)} - c^{(j)})$, and $w^{(j)}$, $T^{(j)}$, $d^{(j)}$ and $c^{(j)}$ are the weight, tardiness, due date and completion time of job *j*, respectively. It is easy to see that different sequences provide different weighted tardiness. We seek for a scheduling scheme that minimizes the value.

The structure of an assembly shop can be defined by its job structure and machine structure as follows.

Job Structure

In the problem, the arrival time, $r^{(j)}$, due date, $d^{(j)}$, and priority, $w^{(j)}$, of jobs are known in advance. The processing routing is predetermined and some assembly operations may present. Due to the assembly structure, jobs may revisit the same machine more than once (reentrance).

The processing of the job on the machine is called *operation*. Each operation, say operation *i*, has the processing time of p_i and a set of preceding operations, $Preceed_i = \{i' : i' \text{ is the operation immediately preceding } i \text{ in the routing}\}$ and succeeding operations, $Succeed_i$. They are determined from the job route. Operation *i* need to be processed by workstation, wk_i, and the required machine status is $status_i$.

Machine Structure

The machines on the shop are grouped into workstations according to their capabilities. In general, machines doing the same type of tasks will be assigned to the same workstation. Workstation may consist of a single machine or multiple machines working in parallel.

in parallel. There are three types of parallel machine workstation. The first type is identical parallel machine workstation where all machines in the workstation are exactly the same. The second type is parallel machine workstation with different speeds. The actual processing time of operation i is determined from $\frac{p_i}{f_k}$, where f_k is the speed of machine k referencing to the average machine. The last type is the unrelated machines in parallel where the actual processing time of operation i may be different on each machine.

Machine Setup

Some machines require special setup before processing a particular task. The machine setup time depends on the current machine setting, $status_i$ and the new setting, $status_{i'}$. The setup time can be looked up from the setup matrix. For example, a machine is in status "B" after cutting a 4" steel tube. It needs 10 minutes to adjust the machine to status "C" that is required to cut 1' steel tube.

Batch Machine

There is another type of machines that is able to process multiple tasks in a single run. This type of machines is called *batch machine*. We consider two types of batch machine workstations. The first type is the workstation that processes single type of tasks (single family). The processing time of this batch machine is fixed, $p_i = p$. Setup time is not required. The other type of batch workstations is the single batch machine workstation that processes various types of tasks. The tasks are separated into *families*. Tasks from different families cannot be mixed in the same batch. After the machine complete a batch, it may require setup to prepare the machine to process a new batch if the tasks in those two batches are in different family.

In this dissertation, we restrict the processing station types to single machine, parallel machine with different speeds and single batch machine. Therefore, the detailed list of workstations is as follows.

Single machine workstation:

- Single machine workstation without sequence dependent setup
- Single machine workstation with sequence dependent setup time

Parallel machine workstation:

- Parallel identical machine workstation without dependent setup time
- Parallel identical machine workstation with dependent setup time
- Parallel related machine workstation without dependent setup time
- Parallel related machine workstation without dependent setup time

Batch machine workstation:

- Batch machine workstation with single family
- Batch machine workstation with multiple families

1.2 Problem Statement

It is well known that traditional MRP systems lack the ability to dispatch and schedule jobs in an assembly shop. Ad hoc priority rules are normally used to overcome this incapability. Though there are a number of priority rules, most of them provide solutions that are far from optimal. We intend to develop a series of heuristics that are more effective than priority rules in dispatching and scheduling jobs in an assembly shop environment. Currently, the majority of the research on advanced job-shop scheduling is limited to theoretical problems. They cannot be easily applied to the problems where assembly operation, jobs release time, multiple type of workstations are considered.

Furthermore, there is a lack of good user interfaces in the scheduling system. Most of the past research has developed computer codes that are lacking in usability. We intend to develop a system that can link the current research in scheduling to industry users and simplify the heuristic development process. This system can be shared among the researchers and industrial schedulers. Thus, this research contains both theoretical and practical aspects.

1.3 Research Objectives

1. Propose new heuristics for solving the assembly type job-shop problems. Traditional dispatching rules are the widely used in practice. They are simple to implement and provide satisfactory results. However, there is a need for better scheduling techniques with higher computation complexity as the computation speed has increased while the computing cost is decreasing. New types of solution techniques such as shifting bottleneck and local search can provide better schedules in a moderate amount of time. The shifting bottleneck heuristic will be explored in this research. This decomposition concept is applied to other objectives and manufacturing environments. The purpose is to develop an efficient heuristic for assembly job shop scheduling that minimizes the weighted tardiness.

2. Develop sequencing and scheduling algorithms for the sub-problem required in 1. Specifically, it is developed for single machine scheduling with tails.
3. Extend the above development to two new sub-problems. These problems are parallel machine scheduling problem with tails, and batch machine scheduling problem with tails.
4. Extend the above heuristics for other objectives including minimize weighted flow time, maximum weighted lateness, makespan, etc.
5. Develop a computerized scheduling system with proficient user interface. This research is trying to fill in the gap between the scheduling theories and industry implementations.

1.4 Significance of the Research

The significance of the research is two folds. First, there is a lack in efficient dispatching and scheduling techniques for assembly shop. The development of the heuristic along with sub-problems optimization will be the theoretical contributions. Second, the development of scheduling system with good user interface will be the contributions to industries who need forefront dispatching and scheduling scheme and researchers who need a tool to demonstrate the performance of their newly developed heuristic and to compare the results with others.

1.5 Dissertation Organization

The next two chapters are on literature reviews with chapter 3 focusing on model development and shifting bottleneck decomposition. We discuss various methods on solving scheduling problem in job-shop environment, and select shifting bottleneck as the rudimentary concept due to its computational complexity and flexibility. All sub-problems and aggregation method are discussed in details. Chapter 4 is devoted to the first sub-problem, single machine scheduling with tails where tails are post processing tasks. We develop a priority rule and a sequence improvement procedure based on critical path analysis. The random search is added to the heuristic when dependent setup time is included.

In order to develop a more generalized problem for assembly shop, we need to solve two other sub-problems. This is done in chapter 5 and 6. The first sub-problem is scheduling parallel machines with tails. We propose two methods. One is based on an extension of the results from chapter 4. The other is based on beam search. Beam search is enhanced by adding random search steps on evaluating step when dependent setup time is considered. The second sub-problem is scheduling batch machine with tails. We propose two heuristics. One is based on priority rule and dynamic programming. First, we generate a full-batch sequence using a priority rule. This sequence is improved by checking whether smaller batch size can improve the objective value using dynamic programming technique. The next heuristic is developed for batch machine scheduling where multiple family of tasks are concerned. The tasks from different families cannot be processed on the same batch, and there exists a setup time when switching from processing tasks in one family to another. A new version of shifting bottleneck is developed from the results in chapter 3, 4, 5, and 6. It is further discussed in chapter 7.

Due to the flexibility of shifting bottleneck concept, the extension of the heuristic to other objectives can be achieved by modifying the sub-problems accordingly. Since the single machine sub-problem is studied based on critical path analysis, it can be applied to other objectives such as minimizing weighted completion time (min $\Sigma w_j C_j$), minimizing the maximum weighted lateness (min $wL_{\max}$), and minimizing the makespan (min $C_{\max}$), etc. Changing the evaluation function in the beam search will enable the parallel machine scheduling heuristic to provide sequences for other objectives. The modification of the heuristic for batch machine scheduling problem is based on generating the feasible sequence in the first step. These modifications are discussed in chapter 8.

Chapter 9 presents the design and development of the scheduling system. We apply object oriented programming methodology for the maintainability and reusability of codes. The system developed allows researchers to add and connect their newly developed heuristics easily. They can, then, visualize their results in graphical format and compare them with others. The system also provides scheduling heuristic development tools. Customized heuristic can be developed faster by using the library of codes.

The conclusion of the dissertation is in chapter 10. We also discuss the future extension of the research in the chapter.

CHAPTER 2

LITERATURE REVIEWS

This chapter provides an extensive review on the assembly scheduling problem. As the problem is an extension of classical job-shop problem, we provide a review on job-shop and flow shop scheduling problems. We discuss various methods on solving the problem and make comparisons. A short review on scheduling system is also given.

In the following sections, we discuss the assembly operation, disjunctive graph method which is a well known tool in scheduling, the modification of the graph, and the shifting bottleneck heuristic.

2.1 Job Shop and Flow Shop Scheduling

In classical job-shop scheduling problem, there are *n* jobs that need to be processed on *m* machines. Each job consists of a series of operations, which are excluded under the following constraints.

a) The processing sequence of each job is predefined.

b) The jobs must visit every machine in the shop. They cannot re-visit the same machine more than once.

c) A machine can process only one job at a time.

The objective that is usually of interest is to minimize the completion time of the last job, known as makespan, $C_{\max}$. This objective is comparable to the maximization of the shop utilization. The problem is a well known problem that can be found in most scheduling text books (Pinedo, 1995).

Problem Formulation

Before formulating the problem mathematically, two dummy operations are added to set of operations, N. Operation "0" is the "start" operation while operation "n" is the "finish" operation. The processing times of these two dummy operations are zero. The "start" operation is the first operation to process. After the "first" operation is finished (at time 0), other operations can start. The "finish" operation is done after all other operations are finished. The problem can be described mathematically as follows.

(1) Min t_n ,

subject to

(2) $t_j - t_i \geq p_i$ $\quad (i,j) \in A,$

(3) $t_i \geq 0$ $\quad i \in N,$

(4) $t_j - t_i \geq p_i$ or $t_i - t_j \geq p_i$ $\quad i,j \in E_k , k \in M,$

where,

t_i	= starting time of operation i,
p_i	= processing time of operation i,
A	= set of pairs of operations constrained by precedence relations,
N	= set of operations,
E_k	= set of pairs of operations to be performed on machine k,
M	= set of machines.

The first equation, (1), states the objective of the problem. Because two operations :annot be processed at the same time, constraint (2) says that the difference in finishing ime of two consecutive operations in the same job must be greater or equal to the processing time of the leading operation. Constraint (4) affirms that the difference of finishing time of any two operations must be at least the processing time of the leading operation when considering the operations that need to be processed on the same machine.

Flow shop is another type of production system that is widely studied. It can be considered as a special type of job shop where all the jobs have the same routing. Jobs are processed through a number of stages in series. The extension of flow shop to *flexible flow shop*, includes parallel machines at each stage. Flow shops can be easily found in the manufacturing facilities with high production rate of comparable products. The mathematical formulation of this problem is similar to the one presented previously. The difference is on the set of operations pairs constrained by precedence relations. Therefore, the scheduling heuristic developed for job shop will be able to use for the flow shop and vice versa.

2.2 Scheduling Techniques

Scheduling problems are one of the topics in combinatorial optimization. This problem appears in many areas. Operations researchers normally perceive the problem as a network flow and/or integer programming problem. Computer science people, on the other hand, often think of heuristics such as simulated annealing or genetic algorithms. The artificial intelligence community looks at it as constraint satisfaction issue. From the mplementation perspective, most combinatorial optimization problems are NP-hard which no polynomial time algorithms have been found (Garey and Johnson 1979; Papadimtriou 1994). The complexity of the problem grows exponentially to the problem size. For example, there are $(3!)^3 = 216$ alternatives to schedule 3 jobs on 3 machines flow shop. When the problem of 10 jobs on 10 machines is considered, the alternatives increase to $(10!)^{10} = 3.959 * 10^{65}$ (note: 1 year = $3.156 * 10^7$ seconds). It is almost impossible to solve the problem optimally within a life time.

Blazewicz *et al.* (1991) provide an extensive coverage on mathematical programming formulations for machine scheduling problems. Currently, there are five major classes of techniques in solving them as follows.

2.2.1 Complete Enumeration

Scheduling problem can be formulated as mixed integer programming. Branch & bound method can be used to find the solution. However, the calculations are intractable if the problem size is large since job shop scheduling problem is NP-hard. The majority of researches in this class is on finding a good lower bound by relaxing some constraints. The tight lower bound can eliminate the number of branching drastically. It results in lower the computational time. The works in this area include ones by Ashour and Hiremath (1973), McMahon and Florian in 1975, Lageweg *et al.* (1977), Fisher *et al.* in 1983, Lageweg in 1984, and Carlier and Pinson in 1989.

Beside reducing the computational time, the lower bound can be used to check whether a schedule is at optimal (Carlier and Pinson 1989). The methods in this class are sensitive to particular problem instances.

2.2.2 Heuristics

Due to the complexity of the problem, optimal solution may not be the main interest. Heuristics are developed to find an acceptable solution. Some well-known heuristics for solving the problem are discussed below.

2.2.2.1 Beam Search: This method is close to branch & bound method. Instead of branching on every node, it skips some nodes that are not promising. Lowerre (1976) was the very first researcher in this area who applied this technique on speech recognition. Fox (1983) and Ow & Morton (1988) applied this technique on solving the scheduling problem. They reported that it outperforms priority rules.

2.2.2.2 Priority Rules: There are various types of priority rules such as Earliest Due Date (EDD), Shortest Processing Time (SPT), First Come First Serve (FCFS), Minimum Slack Time (MST), Earliest Operation Due Date (EDD-O), etc. The priority rule technique schedules the jobs according to some indices. These indices are determined from the jobs' or machines' characteristics. It might be as simple as SPT rule where the indices are assigned according to the processing times. It might be as complex as ATC (Apparent Tardiness Cost) rule where parameters are needed to be predetermined. If the information on due date is used, there are three main types of approaches -- allowance-based, slack-based, and ratio-based priorities. The algorithm by Giffler and Thompson (1960) can be considered as a basis for other priority rule based heuristics on job shop scheduling.

Baker (1984) and Vepsalainen & Morton (1987) conducted surveys on sequencing rules with tardiness oriented in job shop. Baker, also, discussed the factors that affected the performance. The interested readers can find a good survey on priority rules organized by Blackstone, Phillips, and Hogg in 1982. Some major priority rules and their operating environments are exhibited in Table 2-1. Priority rule method is simple and fast; therefore, it is widely used in practice. However, quality of the solutions might not be so honorable. The solution might be far from optimal.

Table 2-1: Some elementary priority rules (Pinedo 1995)

	Rule	Data	Environment
1	SIRO, service in random order	-	-
2	ERD, earliest release date first	r_j	$1 \mid r_j \mid Var(\Sigma(C_j - r_j)/n)$
3	EDD, earliest due date first	d_j	$1 \mid\mid L_{\max}$
4	MS, minimum slack	d_j	$1 \mid\mid L_{\max}$
5	SPT, shortest processing time first	p_j	$Pm \mid\mid \Sigma c_j$ $Fm \mid p_{ij} = p_j \mid \Sigma C_j$
6	WSPT, weighted shortest processing time first	w_j, p_j	$Pm \mid\mid \Sigma w_j C_j$
7	LPT, longest processing time first	p_j	$Pm \mid\mid C_{\max}$
8	SPT-LPT,	p_j	$Fm \mid block, p_{ij} = p_j \mid C_{\max}$
9	CP, critical path	$p_j, prec$	$Pm \mid prec \mid C_{\max}$
10	LNS, largest number of successors first	$p_j, prec$	$Pm \mid prec \mid C_{\max}$
11	SST, shortest setup time first	s_{jk}	$1 \mid s_{jk} \mid C_{\max}$
12	LFJ, least flexible job first	M_j	$P_m \mid M_j \mid C_{\max}$
13	LAPT, longest alternate processing time first	p_{ij}	$O2 \mid\mid C_{\max}$
14	SQ, shortest queue first	-	$P_m \mid\mid \Sigma C_j$
15	SQNO, shortest queue at the next operation	-	$J_m \mid\mid \gamma$

2.2.2.3 Shifting Bottleneck (SB): This method decomposes the problem into a number of one machine scheduling sub-problems. It sequences the machine one after another until all machines are sequenced. First, it finds the bottleneck machine which is determined from the decomposed sub-problems. The sequence is determined for that machine. Among the unsequenced machines, it determines the next bottleneck machine. That machine is scheduled next. The sequence of scheduled machines are re-optimized. Then, a new set of decomposed problems is constructed for the unsequenced machines. The bottleneck machine is determined and sequenced. The method iterates until all the machines are sequenced. In 1988, Adams, Balas, and Zawack purposed the heuristic to solve the classical job-shop scheduling. There are some variations of Shifting Bottleneck heuristic done on other objectives (Uzsoy, Lee, and Martin-Vega, 1992; Pinedo and Singer, 1995).

2.2.3 Local Search (Neighborhood Search)

Local search technique is considered the most recent method in solving combinatorial optimization problem. It is based on random search technique. Though there are a number of local search techniques, they are all based on four aspects of design.

(i) Mapping of the solution instance to the algorithm

(ii) Neighborhood design

(iii) Search technique

(iv) Acceptance-rejection criterion

A good review on Local search methods can be found in Anderson *et al.* (1995). Three major types of neighborhood search -- tabu search, simulated annealing, and genetic algorithm are reviewed.

2.2.3.1 Tabu Search: This technique was first proposed by Glover (1977) in solving nonlinear covering problems. It has achieved impressive practical successes in applications such as scheduling, computer channel balancing, cluster analysis, space planning, etc. (Glover 1986, 1987; Glover *et al.* 1985; Glover and McMillan 1986). Tabu search constrains the search by forbidding some moves (tabu). These forbidden moves are freed after a period of time. Tabu search, unlike hill climbing heuristic, guides the search to continue when no improving move is found. It prevents the move from falling back to the local optimum that has just visited. The detailed discussion of the fundamental Tabu search can be found in Glover (1989, 1990).

The performance of the Tabu search is rather impressive. Widmer and Hertz (1987) compared tabu search with six other approaches in flow shop sequencing problem with 20 jobs and 20 machines. The computing cutoff time was twelve minutes on IBM-PC. They found that tabu search provides better solutions for 80% of the cases. Taillard (1989) applied this technique to job shop scheduling. He also reported that the technique performed better than shifting bottleneck heuristic (Adams *et al.* 1988) and simulated annealing (Van Laarhoven *et al.* 1992) in term of solution quality and computational effort on a set of ten jobs on ten machines problems. Barnes and Chambers (1992) included seven dispatching rules in finding the starting point. They encountered a premature termination where all the moves were tabu. It was solved, simply, by clearing the tabu list. Dell'Amico and Trubian (1993) focused on neighborhood structures of job shop scheduling problems. They developed a more complex search and compared five types of the neighborhood structures. Widmer (1991) tested this technique to job shop scheduling problem with tooling constraints. Tabu search was extended to tackle multiple machines job shop scheduling problems by Dauzere and Paulli (1994). Some of single machine scheduling problems (Laguna et al. 1991, 1992; Woodruff and Spearman 1992) and multiple machines scheduling problems (Barnes and Laguna 1992; Laguna and Gonzalez-Velarde 1991) were solved efficiently with this technique.

2.2.3.2 Simulated Annealing (SA): This technique simulates (physical) metal annealing process. A feasible starting point is first selected. Then, ε's (a small variation) is added to it to find a neighbor point. If the neighbor point has the objective values lower than its parent, accept it as a new parent. Otherwise, accept it with probability p where p is determined by a function of control parameter (temperature). When the temperature is high, p is also high. The temperature is reduced as the search continues. This allows the search to jump out before sticking to a local minimum (maximum). The heuristic was, independently, introduced by Kirkpatrick et al. (1983) and Cerny (1985). It is easy to implement but requires high computation. When modeled as a Markov chain (Aarts and Van Laarhoven 1985; Lundy and Mees 1986; Romeo and Sangiovanni-Vincentelli 1985), it can show that the global optimum will be reached as the control parameter converges to zero (Van Laarhoven *et al.* 1992; Lundy and Mees 1986).

The application of SA in job shop scheduling (Van Laarhoven *et al.* 1992; Matsuo *et al.* 1988) shows that the algorithm can provide quality solutions comparable to or better than other tailored heuristic such as shifting bottleneck (Adams *et al.* 1988). It requires relatively little insight into the problem structure. However, the computation time is higher.

2.2.3.3 Genetic Algorithm (GA): GA tries to simulate natural evolution. First, the feasible solution needs to be represented in chromosome form (string of values). Each chromosome will have a fitness index according to the objective function. The chromosomes can be cross-over with others with some probability hoping that good genes from parents might come to their children. The chromosome can be mutated with probability q. Thus, the search will not get stuck at a local minimum. The chance that the child can live will be according to its fitness index. To make the computation possible, the size of population will be limited. The parents will die out with probability d. This method starts from a given number of populations. Three major operators including reproduction, crossover and mutation guide randomly generated solutions towards high-quality solutions (Goldberg 1989; Davis 1991). GA is based on the theory of evolution (Rechenberg 1973, Holland 1975, Schwefel 1977).

The examples of applying GA to job shop scheduling can be found in Yamada and Nakano (1992) and Croce *et al.* (1995). However, GA algorithm alone may not provide a good performance on job shop scheduling problem (Dorndorf and Pesch 1995). The incorporation of GA with other heuristics such as local searches (Davis 1985; Whitley *et al.* 1989, Husbands *et al.* 1991; Starkweather *et al.* 1992; and Hilliard and Lipeins 1988), priority rules (Dorndorf and Pesch 1995), Tabu search (Glover *et al.* 1995) or shifting bottleneck (Dorndorf and Pesch 1995) can provide the performance improvement.

2.2.4 Simulated Neural Networks (SNN)

Hopfield & Tank (1985) neural structure & methodology is adopted as an optimization tool. When the Hopfield network is perturbed (increase the energy), it tries to find a new minimum energy point. The energy level of the network represents the objective value. The optimization can be done by formulating the structure of network according to the problem. There are various structures of the networks applied to solve scheduling problem (Satake *et al.*, 1994; Arizona *et al.*, 1992; Lo and Bavarian, 1991; Foo, and Takefugi, 1988a; 1988b; 1988c; Zhou *et al.*, 1990; 1991; Van Hulle *et al.*, 1991a; 1991b; Johnson and Adorf, 1992). However, Hopfield based networks cannot guarantee to find the optimal solutions. It might not even find a feasible solution. To avoid being trapped in a local minimum, a stochastic network may be used (Arizona *et al.*, 1992). This method is not considered as an artificial intelligence in computer science point of view.

Another application of SNN is to be used as a learning mechanism. The back propagation structure provide the learning ability. After the network is trained with several examples, it starts to know how to make the appropriate decision (Sabuncuoglu and Hommertzheim, 1992; Hayes and Sayegh, 1992; Chryssolouris *et al.*, 1991). This type of network is broadly used as a dispatching rule selection for the scheduling system (Cho and Wysk, 1994; Pierreval, 1993). It might be combined with expert systems (Rabello and Alptekin, 1989; Rabello *et al.*, 1993; Sim *et al.*, 1994), ATC rule (determine the parameters) (Kim and Lee, 1993) or markov model (Yih *et al.*, 1993).

2.2.5 Artificial Intelligent

Expert systems (ES) are widely used as a scheduling tool in practice. Its goal is to consistently duplicate the results of a human expert (Reinschmidt *et al.*, 1990). The knowledge (rules, frames) is extracted from human experiences. Expert systems are able to provide the solution promptly. However, the results might be far from optimal (Fox, and Smith, 1984). ES is suitable for a complex system that is hard to model mathematically. The implementing system at Westinghouse plant shows a tremendous saving (Miller, Lufg, and Walker, 1988). Another promising method in AI area on job shop scheduling is constraint satisfaction. It is suitable for scheduling generally entail large search spaces with hundreds or even thousands of variables, each with hundreds or thousands of possible values. The technique aims at reducing the effective size of the search space to be explored in order to find a satisfactory solution (Sadeh and Fox, 1996; Sadeh *et al.*, 1995).

2.3 Comparison of the Techniques

Γhe expert system and priority rules provide the lowest quality of solution but they are ;imple and fast. Branch and bound method requires very high computation time but it ;uarantees to find the optimal solution. Between these methods lies SB and neighborhood earch techniques. We drop the neural network technique from out consideration due to he costly computation time. Until analog computer is fully developed, this technique will ot be able to compete with SB or neighborhood searches.

Neighborhood search techniques can provide very high quality of solution. In ıeory, the global optimal should be found at one point. However, it might take a life time. Among three types of neighborhood searches, TB requires the least computation time to achieve the similar quality of solution. It follows by SA and, then, GA. SA seems to be the simplest method which can be applied to various types of problems without major modifications. SA and TB are claimed to be a part of GA.

SB is the only technique particularly developed for scheduling problems. It is based on decomposition technique. The problem is decomposed to sub-problems. The sub-problems can be solved with different techniques depending on their structures. This gives a flexibility to apply the heuristic on complex problems. Unlike neighborhood search, there is no theoretical proof that SB will find the optimal solution.

For assembly shop problem, there are multiple types of workstations. It is very difficult to design a tabu list structure that can efficiently direct the search toward the optimal solution. As GA and SA tend to require high computation time, our research will be based on extending Shifting Bottleneck concept to the assembly shop scheduling problem.

Computational Time

The data used to plot Figure 2-1 and Figure 2-2 came from Laarhoven *et al.* (1992), Croce *et al.* (1995) and Dell'Amico and Trubian (1993). Different types of computers are used. Therefore, the trend of the curve in Figure 2-1 should be focused, not the value. Figure 2-1 shows that the computation time for SA is exponentially growth to the problem size. When the problem is double in size (from 100 operations to 200 operations), the computation time grows more than 40 times (Figure 2-2). In the mean time, the computation time of SB grows at about the same speed as the problem size (about 2 times).

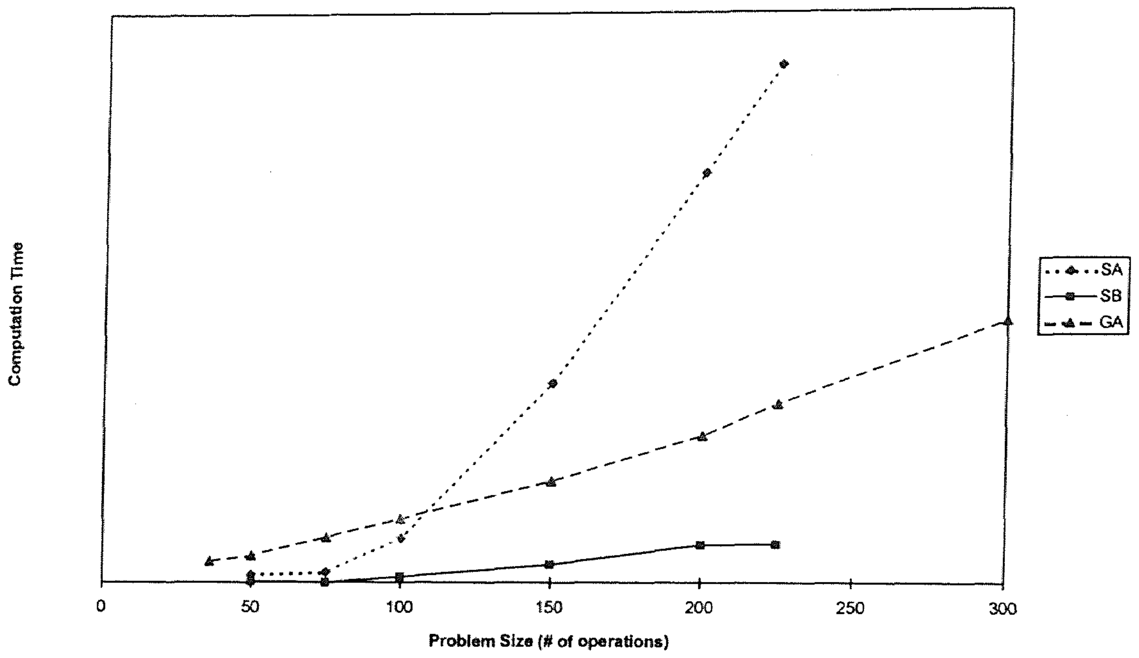


Figure 2-1: Computation time vs. problem size

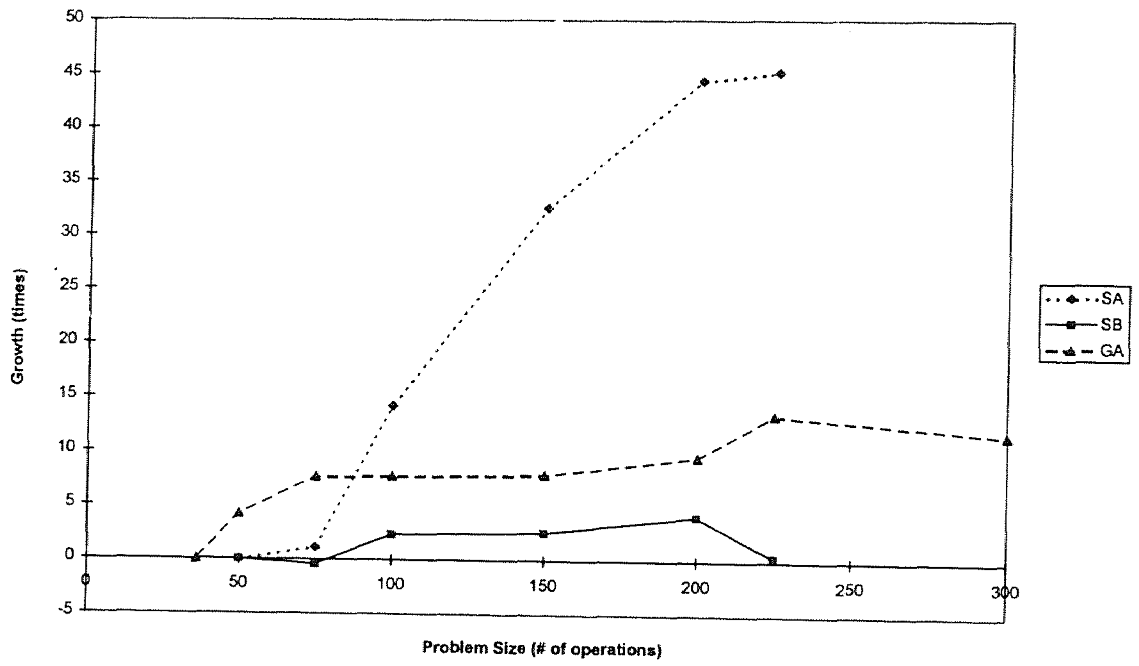


Figure 2-2: Computational growth vs. problem size

2.4 Scheduling Systems

The fundamental of scheduling system consists of three major modules -- database and knowledge-base, schedule generation and re-generation, and user interface modules (Pinedo 1995). These three modules play a crucial role in the functionality of the system. A nice review and discussion can be found in the book by Pinedo (1995). In the following, we provide some flavors on this topic.

Decision Support Systems (DSS) in Scheduling

Decision Support Systems (DSS) are defined as interactive computer based systems that help decision makers utilize data and models to solve unstructured problems. The systems provide simple interfaces for non-computer people to use interactively. They emphasize the flexibility and adaptability to accommodate changes in the environment and decision-making approach. DSS tends to aim at the less structured, under-specified problem that upper-level managers typically face. Models and analytic techniques with traditional data access and retrieval functions are frequently being applied (Sprague & Carlson, 1982).

The word DSS is used vaguely to the system that provide "intuitive validity". Most of the scheduling systems could claim that they are DSS. The following is an example of the PC-based DSS system developed by Castillo (1992).

The system was developed based on network schedule representation and specific scheduling optimization algorithms. The system contains graphical user interface (GUI). User draws the network describing the scheduling problem similarly to project planning software. The system was implemented at a major pharmaceutical firm in Mexico City, Mexico. The company used a well-known MRP package for planning purposes. The production scheduling department was using "Microsoft Project" on a PC to tackle the problems on day-to-day operations. This software provided database and critical-path analysis capabilities, but did not include specific production scheduling algorithms. The new system adopted Balas' algorithm (1969) that finds a critical path in the disjunctive graph in minimizing makespan. The algorithm stops when it reaches the time upper bound and reports the best solution found. It employs dispatching rules such as SPT (Shortest Processing Time), WSPT (Weighted SPT) and DDATE (Due Date Dispatching) for minimizing other performance measures.

Jones *et al.* (1995) indicated that the failure of existing scheduling systems was due to the ignorance of important constraints such as material handling system, incapability to schedule with multiple objectives, difficulty to install & integrate into the pre-existing shop floor control and slowness. A successful scheduling software, OPT (Optimized Production Timetables), developed by Goldratt in the late 1970s was using computerized Kanban method focusing attention on bottleneck operation (Spencer and Cox 1995b). Fry *et al.* (1992) conducted a survey on 60 OPT implementations in the U.S. He found that three firms were no longer using the software, four were implementing and 15 were using OPT. The weaknesses were the unfriendly user interface, the requirement of extremely accurate and timely feedback from the shop floor, sophisticated system, high maintenance costs and the awkward results that were not intuitive to users.

The new area of research is on applying learning mechanism to the scheduling system (Aytug et al. 1994; Shaw et al. 1992; Yih 1990; Shaw and Whinston 1989; and Shaw 1988).

2.5 Assembly Operation

The assembly operation merges multiple components of the same job into a single part. For example, the back rest and the frame must be assembled to form a complete chair. The assembly process can only begin when both the frame and the back rest have been completed. The synchronization may delay the job completion time and create inventory. When all resources are used on producing the frames leaving the production of the back rests far behind, the complete chairs cannot be delivered.

This same structure can be applied to represent the out-sourcing constraints where the production need to wait for parts from suppliers. If the production starts too early, it may have to stand on the factory floor waiting for some components on delivery. Not only the resources are not smartly utilized, but the object also take the valuable space on the shop floor.

Some schedulers may avoid assembly structure by breaking the jobs into a number of sub-jobs. Each sub-job is considered as a job in the previous sense. The synchronization among the sub-jobs is solved by introducing the work-in-process (WIP) buffers through MRP, assigning pseudo deadlines for the sub-jobs (Roman and del Valle, 1996) or apply Just-In-Time (JIT) production system. These three approaches can facilitate the scheduling problem. When enough WIP inventories of the sub-jobs are introduced, the need to complete the sub-jobs within the time limit is subsided. The production is done to replenish the WIP used. The scheduling problem is transformed to an inventory problem. This technique may increase the production cost due to the WIP inventory. For the second technique, determining the deadline for each sub-task will be a new problem. When the deadline of the sub-jobs are set too loose, it might not provide enough time to complete the assembly procedure and its succeeding operations. If the deadlines are too tight, some sub-jobs may be expedited without the real necessity. This may increase the production cost.

By changing the production system to JIT, it can eliminate the scheduling task and WIP inventory. When the demand arrives, the Kan-ban's are passed to trigger the production at various stations. WIP is minimized because the production will occur only when there is a demand. However, there is no clue for the operators on how to select the next job to be processed when there exists a number of Kan-ban cards in the station. Typically, it is done in first-comes-first-serves manner. Therefore, the high priority jobs may be delayed without notices. The system lacks the ability to control the priority of jobs.

The majority of research on assembly operations in job-shop environment is on dispatching and scheduling rule in FMS (Roman and del Valle. 1996; Doctor *et al.*, 1993; Tang *et al.*, 1993; Townsend and Thomas, 1991).

2.6 Disjunctive Graph Method

The job-shop scheduling problem can be transformed to a graph problem (Balas 1969). The disjunctive graph, $G = (N,A,E)$, consists of node set N, conjunctive arc set A, and disjunctive arc set E. The *node* set N corresponds to the set of operations. The precedent relationships between operations on any machine are represented by *conjunctive arcs* (one direction arcs). Machine assignment is represented by the *disjunctive arcs* (double arcs with opposite direction) set. If operations are performed on the same machine, they will have pairs of disjunctive arcs connecting them. The arc weight denotes the processing time. The disjunctive arcs E can be decomposed to v cliques. Each clique, E_k, represents disjunctive arcs pairs on machine k.

In the following example, there are four jobs to be processed on three machines. The first job consists of two operations which need to be done on machine 1 and machine 2 respectively. We named the first and second operations on job 1 as operations 1 and 2. The second job consists of three operations -- operations 3, 4 and 5. Similar to job 2, jobs 3 and 4 comprise of three operations each. All of them have the same processing route. They will be processed on machines 1, 3 and then on machine 2. The details are shown in Figure 2-3 and Figure 2-4.

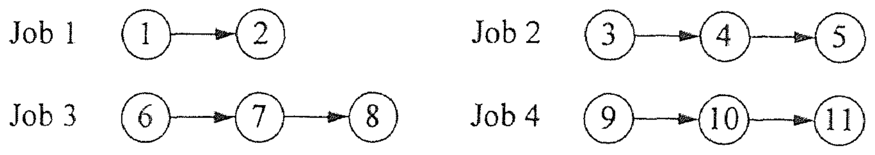


Figure 2-3: Operations in jobs

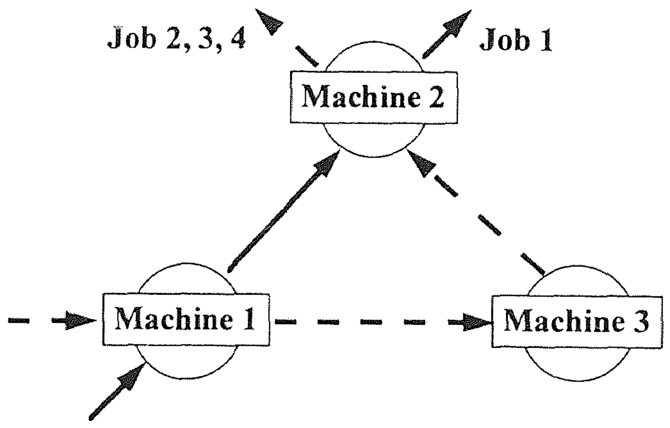


Figure 2-4: Job routing

Figure 2-5 shows the disjunctive graph representation of a problem. The disjunctive arc pairs are displayed as dash-lines with arrows on both ends. Node 0 and 14 are added dummy nodes representing the "source" and "sink" nodes. The conjunctive arcs (solid line with one-sided arrow) connect the operations that have the precedent relationship.

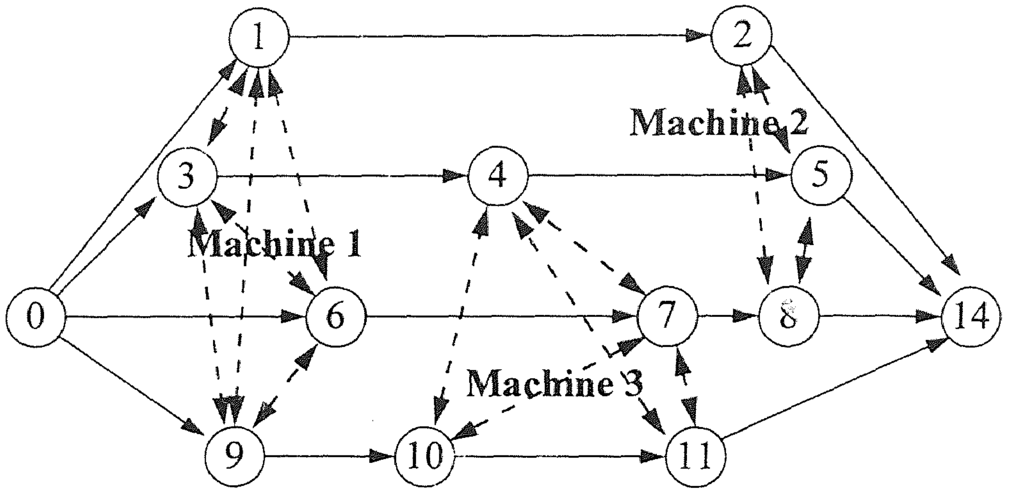


Figure 2-5: Disjunctive graph

A direct graph, *D*(*N*,*A*) is obtained from *G*(*N*,*A*,*E*) by removing the disjunctive arcs (Figure 2-6). A *selection* S_k in E_k is the sub-graph of E_k that replaces each disjunctive arcs pair with a conjunctive arc. Sequencing machine *k* is similar to finding the acyclic selection S_k. Figure 2-7A shows a clique of machine 1, E_1. After selecting conjunctive arcs from the disjunctive arcs pairs in E_1, we obtain S_1 (Figure 2-7B). The sequence of machine 1 is processing operation 1, 9, 6 and, then, operation 3. Figure 2-7C shows a cycle in the selection. A feasible sequence cannot be determined from this selection. The classical job-shop scheduling problem is similar to finding acyclic selections for all the cliques in the graph that minimize the critical path length.

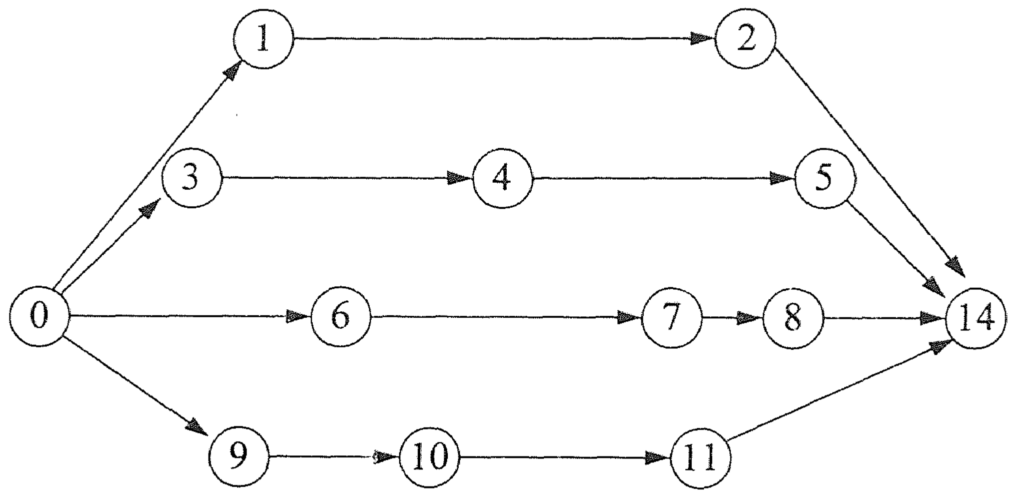


Figure 2-6: Directed graph

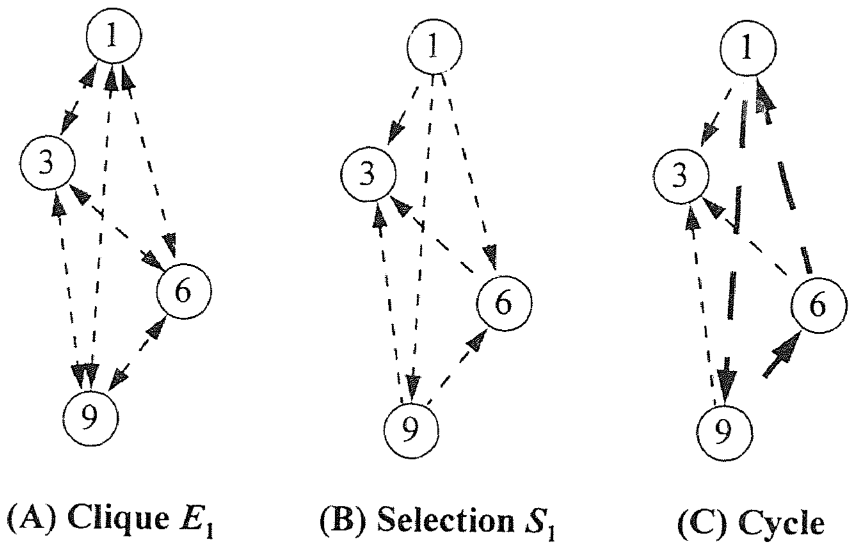


Figure 2-7: Clique and selection

2.6.1 Makespan Determination

The starting time of operations can be determined directly from the graph when complete selections are fixed. Due to the fact that the operation cannot start until all its preceding operations have been completed, the earliest starting time of that operation is the maximum completion time of all the preceding operations. Without cyclic selections, the earliest starting time of the sink can be determined. It is the completion time of the last job or the makespan of the sequence. The Critical Path Methods (CPM) in project management can be applied to the problem (Monks, 1982). The earliest starting time (EST) is the earliest point of time that the operation can start. The latest completion time (LST) is the latest point of time that the operation can start without delaying the project completion time. Both of them can be determined by the following procedure.

i) Calculate the EST of each node.

ii) Set EST of the source node to 0.

iii) Determine the EST of the succeeding nodes.

  (iii-a) Select a node that all the EST of its prior nodes have already determined. Say node *i* is selected.

  (iii-b) EST of node *i* is the maximum of EST of its prior nodes plus the arc length from that node to node *i*.

  (iii-c) Continue until the EST of the sink is determined.

v) Determine the LST from right to left starting from the sink.

  (iv-a) Let LST = EST for all sinks.

(iv-b) Select a node that all the LST of its succeeding nodes have already determined. Say node *j* is selected.

(iv-c) LST of node *j* is the minimum of LST of its succeeding nodes minus the arc length from node *j* to that node.

(iv-d) Continue until the LST of the source is determine. At source, EST = LST = 0.

The operations that contribute directly to the interval of the makespan are called *critical operations*. Delaying any of these operations will cause a delay on the makespan. For the critical operation, EST is equal to LST. There is no room to delay this operation without effecting the makespan. The *critical path* is defined as a chain formed by the critical operations.

The *directed graph*, $D(N,A)$ is the graph $G(N,A,E)$ without the disjunctive arcs. The completion time of this graph can be considered as a lower bound of the makespan as the disjunctive constraint is relaxed. One can regard the $D(N,A)$ as the disjunctive graph for the same problem but with infinite number of machine available.

2.6.2 Modified Disjunctive Graph

Disjunctive graph is an efficient tool to determine the makespan of the shop. It is developed for classical job shop scheduling problem. However, the method can be extended to cover various variations and properties of the shop. Some issues are discussed below.

2.6.2.1 Modified Disjunctive Graph (Due Date): The disjunctive graph discussed before was developed by Balas (1969). It was used as the foundation on developing heuristic in solving classical job shop scheduling problem. As one can see, this graph does not contain the information to determine the completion time of each job. It can only use to determine the completion time of the last job that complete the service (makespan). Uzoy (1992) extended this graph to contain the necessary information for his heuristic where the maximum lateness was minimized. Single sink node is not sufficient to determine the completion time of all the jobs. Graph with *m* jobs requires *m* sinks. Each sink will be assigned to follow the last operation in each job routing. The arcs connecting them have the weight that equal to the operations processing time. Figure 2-8 shows the modified disjunctive graph. The completion time of the jobs can be determined using the similar technique, CPM, as explained earlier. After EST of all the nodes are determined, assign EST to LST for all the sinks. Then, calculate the LST of all the nodes. The EST and LST of the source should be zero. The job completion time is the EST (or LST) of its associated sink node. It is obvious to see that there are at least *m* critical paths in the graph. The length of each path represents the completion time of its associated job.

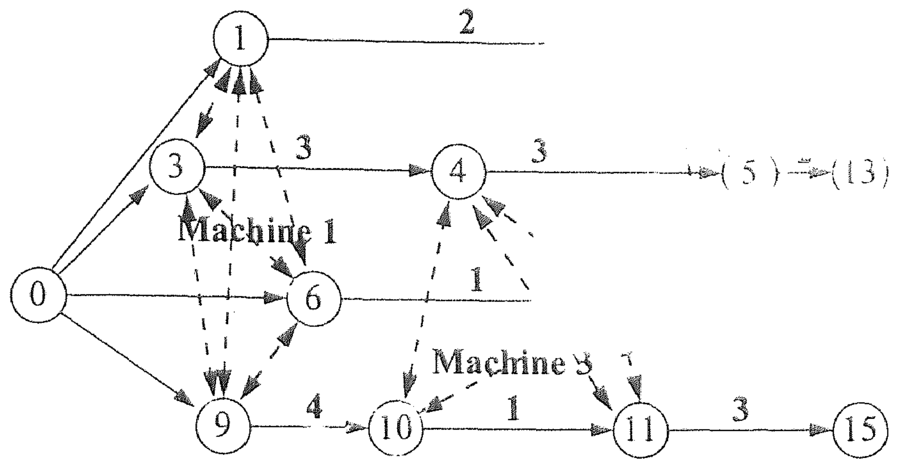
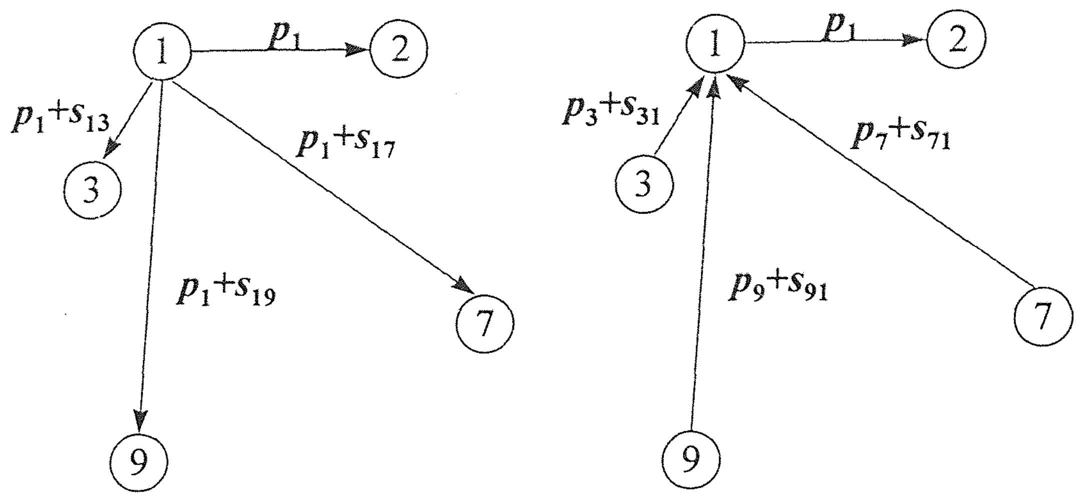


Figure 2-8: Modified disjunctive graph

2.6.2.2 Modified Disjunctive Graph (Dependent Setup Time): The dependent setup time can be incorporated in the disjunctive graph representation by modifying the weights of the disjunctive arc pairs. Each pair can be divided into two types of arcs – outbound and inbound arcs. These arcs represent the time that the machine requires for processing the task and preparing to process a new one. Therefore, the arc length should be the combination of processing time and setup time. Let $a_{ii'}$ represents the arc length from node i to node i'. We obtain $a_{ii'} = p_i + s_{ii'}$ when i and i' are operations in the same clique and $a_{ii'} = p_i$ otherwise. Figure 2-9 shows the modification.



(a) Arc lengths from node 1 **(b) Arc lengths to node 1**

Figure 2-9: Dependent setup time in disjunctive graph

2.6.2.3 Modified Disjunctive Graph (Parallel Machines): For the *m* parallel machine workstation, sequencing the workstation is similar to determine up to *m* selections from the clique associate to the workstation. Each of them should not create a cycle. After the selections are entered in the graph, the job completion time can be determined.

<u>Parallel Machines with Different Speeds</u> : When the speeds of the machines are not equal, further modification is necessary. The outbound arc length from nodes *i* to *j* in the same clique will be changed to $\frac{p_i}{f_k} + s_{ij}$, where f_k is the speed of machine *k* assigned to process operation *i*, and $\frac{p_i}{f_k}$ if *i* and *j* are not in the same clique.

Parallel Unrelated Machines : For the most general case where the processing time of the tasks depends on the processing machine and $\frac{p_{ij}}{p_{ik}}$ is not constant, the outbound arc lengths from node *i* to node *j* in the same clique are modified to $p_{ik} + s_{ij}$ where p_{ik} is the processing time of operation *i* on machine *k*. If *i* and *j* are not in the same clique, the setup time will not be included and the length is p_{ik}.

2.6.2.4 Modified Disjunctive Graph (Assembly Operations): Disjunctive graph formulation can also be modified to represent the assembly structure. Consider Figure 2-10, there are two jobs to be processed. The first job, job 1, consists of five operations. It can be divided into two sub-jobs. Each sub-job can start independently. After they complete the processing, the assembly operation (operation 5) can start. In order to start the assembly process, both sub-jobs must be available at the assembly station. Similarly, there are three sub-jobs for the second job. All three sub-jobs must be completed before the assembly operation, operation 11, can start.

The assembly operation can be modeled by adding conjunctive arcs from the last operations in the sub-jobs to the assembly operation. These conjunctive arcs restrict the assembly operation to start when all the preceding sub-jobs have been completed.

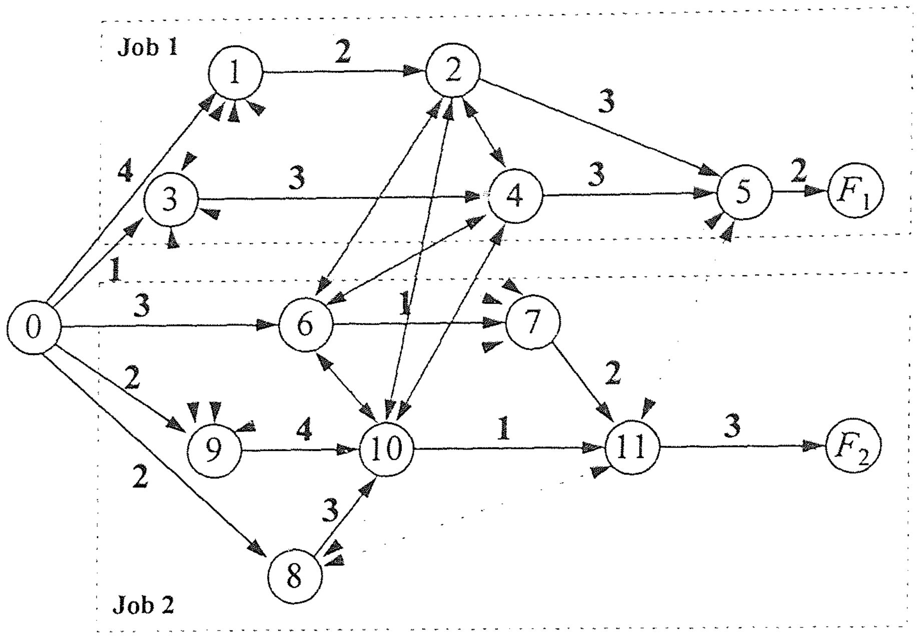


Figure 2-10 : Assembly operations in disjunctive graph

2.6.2.5 Modified Disjunctive Graph (Batch Processing): A special type of machine that can process many jobs in a single operation is called a batch machine. It is frequently found in different industrial processes such as heat treatment, electronic product burn-in process, metal coating, foundry, etc. The batch machine may start after the first job comes. It, also, can accumulate the jobs to its maximum capacity before starting the process. Normally, the processing time of the batch machine is long. It has a high potential to be the bottleneck machine in the factory.

Modeling batch processing is fairly similar to adding assembly-disassembly operation in the graph. All the operations assigned to the same batch must be completed before the batch can start which is comparable with assembly process. After the processing terminates, the processes that follow the job can begin. This is comparable to disassembly operation. To model these properties on the disjunctive graph, dummy nodes are created to represent batches. The operations on the same batch are linked to the dummy node with zero weight. The links from the dummy nodes to the succeeding operations will have the weight equal to the batch processing time. There are links connecting one batch to another according to the sequence. The weight of the link is the batch processing time and the setup time.

Figure 2-11 shows an example of a batch machine in a shop (Figure 2-10). In this example, machine 1 is a batch machine that can process up to 2 parts simultaneously. Operations 1 and 3 are assigned to the first batch while operations 7 and 9 are assigned to the second one. The processing time of batch B_i is P_{B_i}. There is a setup time, $S_{B_iB_{i'}}$, on switching from processing batch i to processing batch i'.

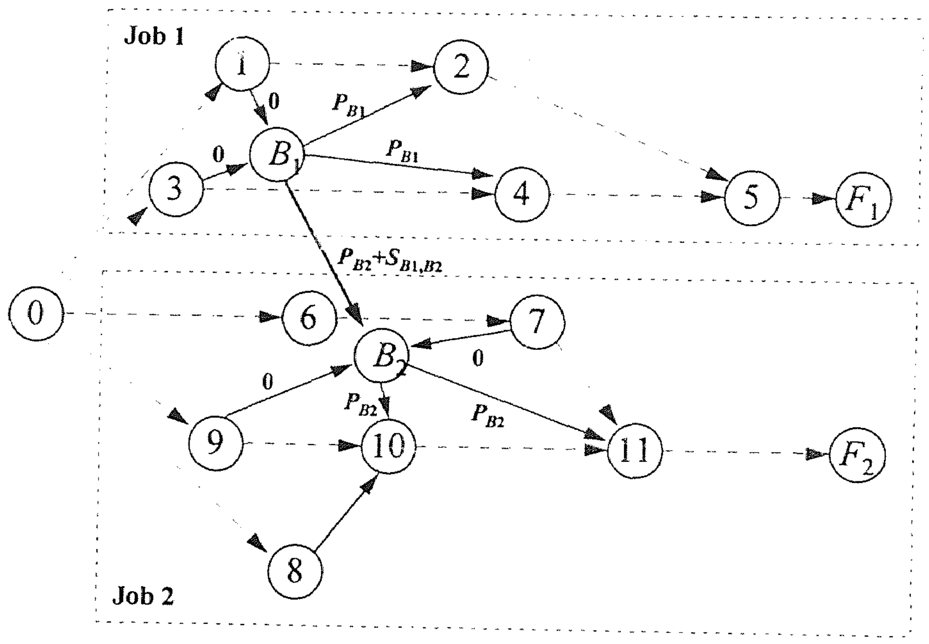


Figure 2-11 : Disjunctive graph with a batch machine

2.6.2.6 Modified Disjunctive Graph (Machine Availability): Machine breakdown or unavailability can also be captured on the disjunctive graph representation. The link from source to a dummy operation will be added. Its weight, r_m, equals to the machine release time or the time when machine will be ready again after the breakdown. The dummy operation is connected to all the operations that are assigned to that particular machine. Their weights are nulls. This method will restrict the processing not to start before time r_m. Figure 2-12 shows that the release time of machine 2, r, is added to the graph.

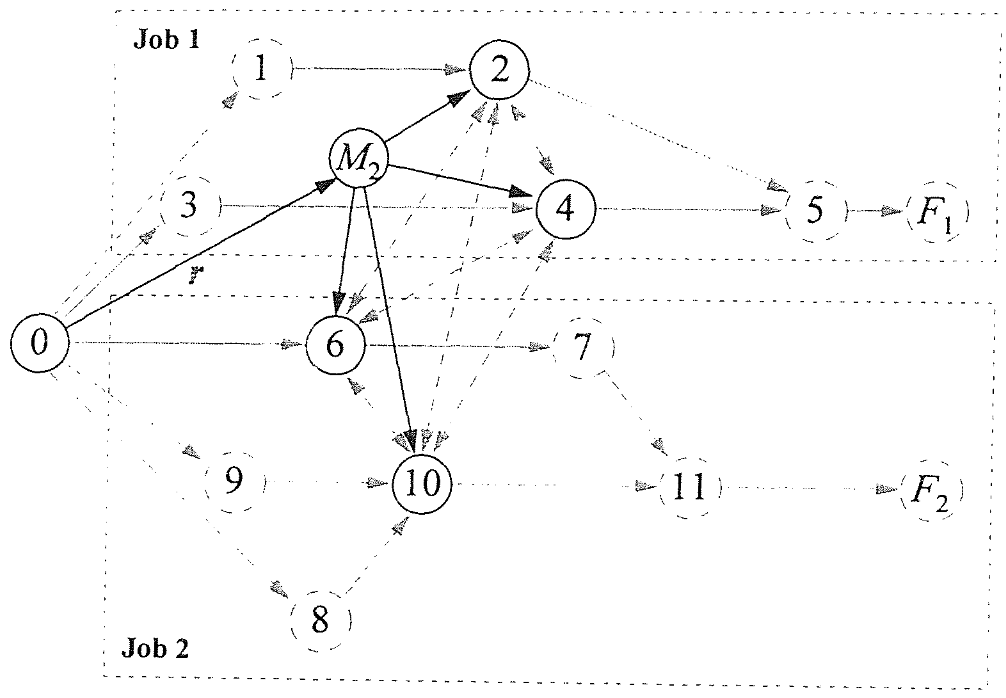


Figure 2-12: Modified disjunctive graph -- machine available time

This modification (Machine Available Time) is necessary when developing the dynamic version from the static scheduling scheme. Suppose that we have fixed the machine sequences for the shop. After time t, some new jobs may arrive. Some operations may have been completed. Some might not have started and others are on processing. Since job preemption is not allowed, we cannot assume that all machines are available at that instance. Machine release time constraint is necessary.

2.6.2.7 Modified Disjunctive Graph (Scheduled Maintenance): The preventive maintenance can be perceived as an extra job. This job has only one operation. The processing time of this operation is the time required for the maintenance. The release time of the job is the expected time to start the maintenance on the machine. The significance of the maintenance can be specified as the job weight. If a high weight is assigned to the job, the time window for the maintenance becomes more rigid. With this method, a number of preventive maintenance can be sequenced on various machines. The disjunctive arc pairs can be added to maintenance operations that require to be processed at the different instance due to the shared resources. Please be reminded that the completion time of this added job should not be counted towards the shop objective value.

2.7 Shifting Bottleneck Heuristic

The shifting bottleneck heuristic was developed by Adam *et al.* (1988). It aimed to find the best sequence for job shops that minimize the shop makespan (referred to ABZ in what follows). This heuristic was reported to find good schedules in a considerable amount of time. Due to the advance in computing technology, computers are faster and cheaper than ever. The need of robust scheduling heuristic emerges. Dispatching rules (one pass heuristics) are fast however they might not provide good quality of solutions.

Branch and bound method can provide the optimal solution, but it almost takes everlastingly time to find the solution regardless of the computing speed when considering a medium/large problem (see NP-Completeness). Shifting bottleneck heuristic was positioned between these two methods. It takes less computation time than branch and bound method but provides better solutions than dispatching rules in most cases.

2.7.1 The Concept

The shifting bottleneck heuristic developed by Adams *et al.* (ABZ) is based on disjunctive graph representation. The graph is used as a tool for developing the decomposed problems. Each clique in the graph is split into a sub-problem. Each sub-problem is formulated as a single machine scheduling problem with job release times and due dates. Their release times and due dates are determined from the original graph.

The heuristic determines the sequence for the machines one after another. On each iteration, the bottleneck machine is identified by solving the decomposed problems. The best sequence among the decomposed problems are compared. The one that provides the highest objective value will be selected. The machine on that sub-problem is said to be the *bottleneck machine.* After the bottleneck machine is identified, its sequence is added to the graph. At this point, some previously sequenced machines are re-scheduled. This step is called *local re-optimization.* At the end of the iteration, the new set of sub-problems are determined. These sub-problems will not include the ones that have already sequenced. The procedure continues until all the machines are scheduled. Then, a *final* *re-optimization* may be employed to further improve the solution. The steps in ABZ heuristic are as follows.

2.7.1.1 Problem Decomposition: On applying shifting bottleneck concept, we focus on single machine, say m_j, in each iteration. The machine constraints on the others that have not been assigned the sequence will be relaxed. We may perceive it as, for unscheduled machines, that there are infinite number of machines instead of single machine to process the operations. On this sub-problem, the goal is to find a sequence for that machine to minimize the makespan. In order to analyze the problem efficiently, we may separate the operations in the network into four groups – operations assigned to m_j, preceding operations to the operations in the first group, succeeding operations, and indecisive operations. Without losing generality, we may combine some operations in the second group and perceive them as the release time constraints for operations in the first group. Operations in the third group can, also, be combined and perceived as post processing operations. They sometimes refer as *tails*. The indecisive operations are the operations those are not assigned to m_j and cannot be said the be preceding or succeeding operations in the first group. The operations in this group may be combined and interpreted as *delay precedent constraints*. The decomposition method is discussed in the next sections.

At the very beginning of the heuristic, a directed graph is created from the job information. Figure 2-13 is the directed graph from the example in section 2-3. The sub-problems are derived from this graph. The number of sub-problems equals to the number of unscheduled workstations. In this case, three sub-problems will be derived. The first sub-problem on machine 1 considers nodes 1, 3, 6, and 9. The EST of these nodes will be derived along with the LST. Here, EST of nodes representing the combined operations of the second group and tails, combined operations of the third group, are converted to LST. For this sub-problem, there are four jobs. The release times and due dates of these jobs are the EST and LST of the nodes.

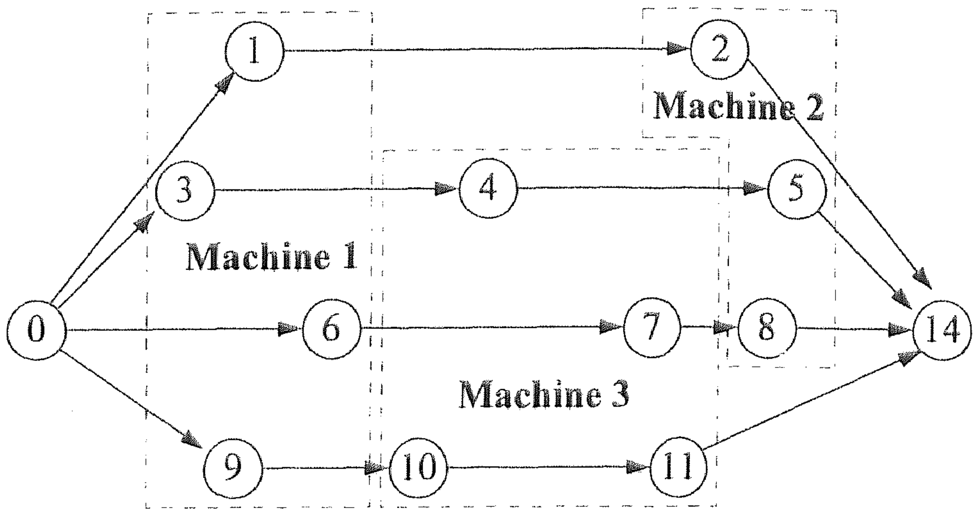


Figure 2-13: Directed graph

2.7.1.2 Bottleneck Determination: SB technique is based on the sequencing intuition that the bottleneck machine should be sequenced first. Then, the less utilized machines are sequenced one after another according to their bottleneck indices. Then, it comes the following question, "how do we identify the machine that is the bottleneck machine and how to sequence it?". This problem was answered by solving the decomposed problems. The sub-problem that provides the highest objective value of the best sequence found is considered the bottleneck machine. ABZ heuristic applied Calier (1982) algorithm on this step. The optimal sequence for the sub-problem is obtained based on branch and bound technique.

2.7.1.3 Sequencing the Bottleneck: Scheduling the machine is equivalent to adding a selection to the graph. The selection is determined by solving the single machine problem. After the bottleneck machine is identified in the previous step, the sequence that provides the minimum objective value on that machine is added to the graph.

2.7.1.4 Local Re-Optimization: Experience shows that this step can improve the quality of the solution. *m* of the scheduled machines are removed from the graph. Then, the removed machines are scheduled back one after another using the same technique in determining the sequence for the bottleneck machine. On doing this, the structure of the decomposed problems are different from the ones that were previously solved. The makespan (C_{max}) (after all the removed machines are sequenced back) may decrease.

After the re-optimization step, the new set of sub-problems is generated. It will not include the machines that have already assigned sequences. The number of sub-problems is reduced. The heuristic continues until all machines have assigned sequences.

2.7.1.5 Final Re-Optimization: After sequences are determined for all machines, the final re-optimization step may be employed. Similar to local re-optimization, the selections are removed from the graph one after another. A new decomposed problem is constructed and solved. The new sequence is put back into the graph. The step continues until there is no further improvement. It is similar to the search for a local minimum.

2.7.2 Related Research

In essence, shifting bottleneck is a technique to decompose a complex problem into sub-problems. Each sub-problem is solved optimally. The aggregation steps are done iteratively. There are some variations of this step when applying to different problems. Most of them are reported to find very good solutions within a reasonable computation time.

Dauzere-Peres and Lasserre (1993) replicated the same model published by Adams *et al.* They found the operation dependency effects from some partial sequences. Therefore, they developed a heuristic to resolve this problem. This modification not only improved the quality of the solution but also reduced the computation time. The problem was further studied by Balas *et al.* (1995).

Uzsoy *et al.* (1992) applied SB to production planning and scheduling problem in the semiconductor industry. Their objective is to minimize the maximum lateness of the jobs. On the similar path, Pinedo and Singer (1995) studied a problem with the objective of minimizing the weighted tardiness objective. The modified ATC (Apparent Tardiness Cost) rule is used in Pinedo and Singer for solving the sub-problems.

In our model, we consider the priority of jobs and their tardiness. The heuristic developed by Adams *et al.* (ABZ) cannot be applied to the new problem. The disjunctive graph cannot handle due date related objectives. Therefore, we adopted the disjunctive graph modification presented by Uszoy *et al.* as explained in the previous chapter. However, their model does not consider the job priorities. A new decomposition method is proposed in this research. The detail of the method is discussed in the next chapter.

CHAPTER 3

ASSEMBLY SHOP MODEL DEVELOPMENT AND DECOMPOSITION

In this chapter, we develop the assembly shop model. As our heuristic is based on the shifting bottleneck decomposition method, the assembly shop problem will be decomposed to appropriate sub-problems. Each of the three sub-problems is discussed in detail.

3.1 The Model

There are *n* jobs to be completed on *m* workstations. Each workstation is a collection of machines performing the same type of works. Each job has to visit a number of workstations and has a predefined route. The processing on a machine is called *operation*. The route may be comprised of assembly operations. In the model, we assign a unique index to each operation. Therefore, we can trace the information on job and processing workstation from the operation index. The workstations can be groups into three categories:

(i) Single machine workstation

(ii) Parallel machine workstation

(iii) Batch machine workstation

In the parallel machines workstation, job can be processed by any machine in the station. Machines in both single machine and parallel machine workstations can process only one job at a time. In contrast, batch machines process jobs in lots. Once the processing on the machine starts, it cannot be stopped until the processing is completed. The remaining unprocessed jobs wait in queues.

Figure 3-1 shows an example of bill of material (BOM) for a chair production. To produce a chair, there are 13 operations as shown in Figure 3-2. These operations are to be performed on six workstations. Of the 13 operations, there are two assembly operations. Operation 12 requires parts from operations 9, 10, and 11, and operation 13 requires parts from operations 5 and 12. We may write the operation precedent relationship as a set, A, as $A = \{(1,2), (2,3), (3,4), (4,5), (5,13), (6,7), (7,8), (8,9), (10,12), (9,12), (11,12), (12,13)\}$.

The shop may have to produce various type of chairs. Each type has its own production routing which can be determined from the BOM. Each job has a fixed due date given by the customer. The precedent constraints set, A, includes all precedent constraints derived from the job routing. The objective of the problem is to find the sequence for all the workstations in the shop that minimizes the weighted tardiness of all obs.

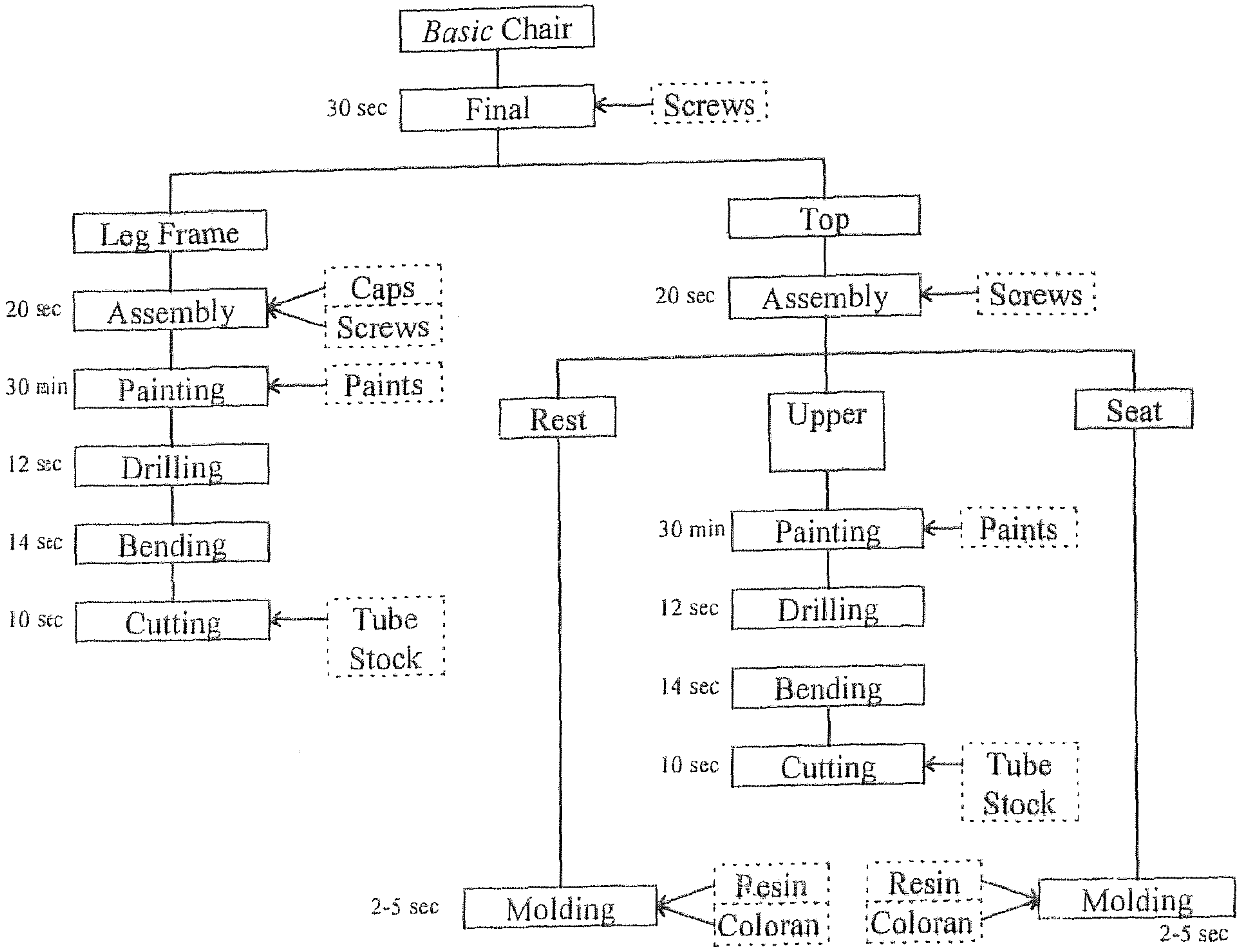


Figure 3-1: Bill of material (BOM)

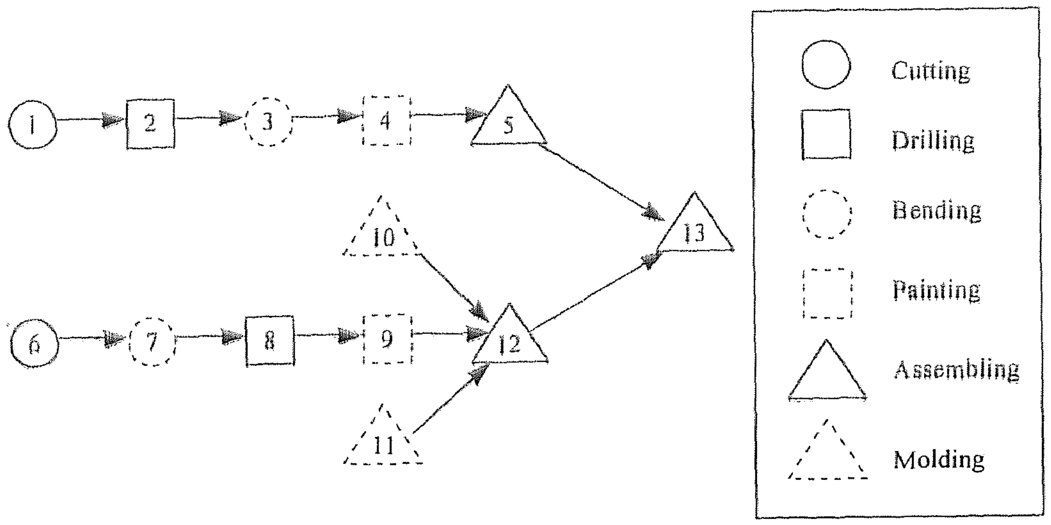


Figure 3-2: Job derived from BOM

Mathematical Model

In this section, we give the mathematical formulation of the basic assembly shop model with three types of workstations. The objective is to minimize the summation of the weighted tardiness of jobs. There are four types of constraints. The first type concerns job routing and release time. The next three types are on machines in workstations including single machine, parallel machines and batch machine workstations. We simplify the model by excluding the dependent setup time. This issue will be addressed later.

(1) $\text{Min} \sum_{j=1}^{n} w^{(j)} T^{(j)}$,

subject to

(2) $t_{i'} - t_i \geq p'_i$, $\qquad (i, i') \in A,$

(3) $C^{(j)} \geq t_i + p'_i$, $\qquad i \in J^{(j)},$

(4) $T^{(j)} \geq C^{(j)}$ and $T^{(j)} \geq 0$, $\qquad j = 1, \ldots, n,$

(5) $t_i \geq r^{(j)}$, $\qquad i \in J^{(j)},$

Single machine workstation

(6) $t_{i'} - t_i \geq p_i$ or $t_i - t_{i'} \geq p_{i'}$, $\qquad i, i' \in E_l, l \in T_1.$

(7) $p'_i = p_i$, $\qquad i \in E_l, l \in T_1.$

Parallel machines workstation

(8) $(x_{ik} \cdot x_{i'k})(t_{i'} - t_i) \geq (x_{ik} \cdot x_{i'k}) p_i$ or

$(x_{ik} \cdot x_{i'k})(t_i - t_{i'}) \geq (x_{ik} \cdot x_{i'k}) p_{i'}$, $\qquad i, i' \in E_l, l \in T_2, k \in M_l.$

(9) $p'_i = p_i$, $\qquad i \in E_l, l \in T_2.$

(10) $\sum_{k \in M_l, l \in T_2} x_{ik} = 1$, $i \in J^{(j)}$.

(11) $x_{ik} = 1$. if operation i is assigned to machine k,

$= 0$, otherwise,

Batch machine workstation

(12) $t_i \geq y_{ib} \cdot tb_b$, $i \in E_l,\ l \in T_{3\cdot}$,

(13) $p_i' \geq y_{ib} \cdot pb_b$, $i \in E_l,\ l \in T_{3\cdot}$,

(14) $pb_b \geq y_{ib} \cdot p_i$, $i \in E_l,\ l \in T_{3\cdot}$,

(15) $tb_{b'} - tb_b \geq pb_b$ or $tb_b - tb_{b'} \geq pb_{b'}$, for all b and b',

(16) $\sum_{i=1}^{m} y_{ib} \leq z$, $i \in E_l,\ l \in T_{3\cdot}$,

(17) $\sum_{b} y_{ib} = 1$, $i \in E_l,\ l \in T_{3\cdot}$,

(18) $yib = 1$ if operation i is assigned to batch b, or

$= 0$ otherwise,

where,

t_i is the starting time of operation i,

A is the set of precedent constraints determined from job routes,

p_i' is the actual processing time of operation i,

$C^{(j)}$ is the completion time of job j,

$J^{(j)}$ is the set of operations belonged to job j,

$T^{(j)}$ is the tardiness of job j,

n	is the number of jobs,
$r^{(j)}$	is the release time of job j,
E_l	is the set of disjunctive arcs (operations) assigned to workstation l,
T_1	is the set of single machine workstations,
p_i	is the processing time of operation i,
T_2	is the set of parallel machine workstations,
M_l	is the set of machines in workstation l,
x_{ik}	= 1 if operation i is assigned to machine k, otherwise 0,
tb_b	is the starting time of batch b,
y_{ib}	= 1 if operation i is assigned to batch b, otherwise 0,
T_3	is the set of batch machine workstations,
pb_b	is the processing time of batch b,
z	is the maximum batch size.

Constraint Interpretation

(1) Objective function

(2) Routing constraints

(3) Job completion time constraints

(4) Determine the job tardiness

(5) Release time constraints

(6) Machine capacity constraints: Only single job can be processed on the machine at any point of time.

(7) Processing time: It is determined from the operation [compare with (13)].

(8) Machine capacity constraints: Only single job can be processed on the machine at any point of time.

(9) Similar to (7) (separated for clarity)

(10) Machine assignment constraints: The operation must be assigned to a machine.

(11) Integer 0/1 variables: They are used to identify the machine assignment.

(12) Batch starting time: Jobs in the same batch must start at the same time.

(13) Processing time: Machine processing time is determined from the batch processing time.

(14) Batch processing time: Batch processing time is determined from the largest processing time of operations in that batch.

(15) Machine capacity constraints: Only single batch can be processed on the batch machine at any point of time.

(16) Batch size constraints

(17) Batch assignment: The operation must be assigned to a batch.

(18) Integer 0/1 variables: They are used to identify the batch assignment.

First, we concentrate on the case that there is only one type of workstation. So, constraints (8) - (18) will be removed. Then, we extend the model to cover three types of workstations. Finally, the dependent setup time constraints (which are not shown in the mathematical model) will be introduced to the model.

3.2 Workstation Decomposition

As our heuristic is based on shifting bottleneck decomposition method (refer to Section 2.7), the original problem will be decomposed to sub-problems. Well formulated sub-problems provide a good solution to the original problem. For instance, incorporating the operations dependency, which was originated from a partial sequence, into the single machine scheduling sub-problem, can significantly improve the quality of the solutions as shown by Dauzere-Peres and Lasserre (1993).

The workstation decomposition is strongly based on disjunctive graph representation (see Section 2.6). In the graph, node represents the corresponding operation; therefore, we may use the word "operation" and "node" interchangeably. In Figure 3-3, there are two jobs that need to be processed on three machines. Operations 1, 3, 7, and 9 are assigned to machine 1. Machine 2 will process operations 2, 4, 6 and 10. The rests, operations 5, 7, 8, and 11, will be processed on machine 3. This disjunctive graph can be decomposed to three sub-problems. Let us focus on one of the sub-problem, say machine 1. On this machine, operations 1, 3, 7 and 9 cannot be started before times 4, 1, 4, and 2 (EST) respectively. If these operations are completed not later than its latest completion time (LCT), both job 1 and job 2 will have the completion time of 11 and 10 which are their lower bounds. If one of the operations is delayed for x units of time, the completion time of the job that contains the operation will be extended for x units of time. The EST and LCT of nodes can be determined by using CPM.

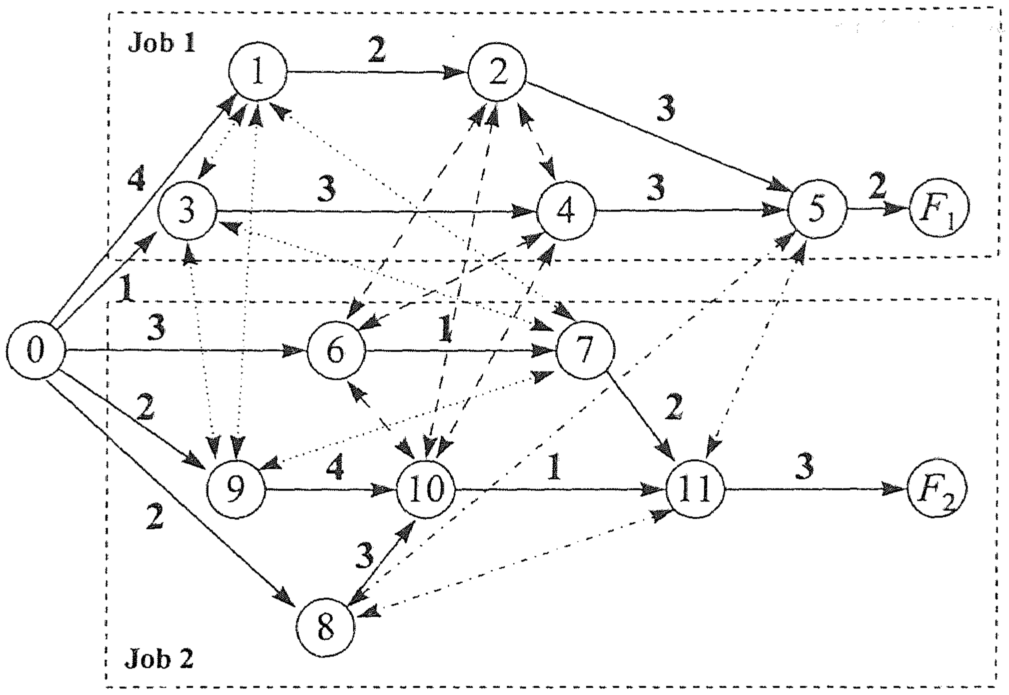


Figure 3-3: Disjunctive graph

Table 3-1: Node information

Node	EST	LST	Remaining process time
1	4	7	5
2	9	9	2
3	1	6	5
4	4	9	2
5	12	12	-
F1	14	14	-
6	8	8	5
7	9	9	3
8	2	4	4
9	2	3	4
10	7	7	3
11	11	11	-
F2	14	14	-

In Figure 3-4, we show the effect of partial sequence to the completion time of the jobs. On this graph, machine 2 has a fixed sequence, {4,10,6,2}. This sequence creates the jobs dependency on determining the sequence of machine 1. Operation 7 cannot be started before operation 3 is completed. This relationship derives from the arcs 3-4-10-6-7. When operation 3 is delayed, it will not only affect the completion of job 1, but it will also affect the completion of job 2. We propose a decomposition method that consider the issue.

The following decomposition approach extracts the information of operations that need to determine the sequence of a particular workstation. The operations assigned to other workstations are removed and replaced by new arcs representing the combined processing times. The method not only determines the precedent relations among the operations due to the fixed sequences on other workstations, but also determines the earliest starting time of the operations as well as the remaining processing time. The new decomposed problem objective is similar to the original problem but only a partial set of operations are considered. We search for the sequence of machines in the particular workstation that minimizes the weighted tardiness of jobs. The graph decomposition algorithm has the complexity of $O(n^2)$ where n is the number of operations in the shop.

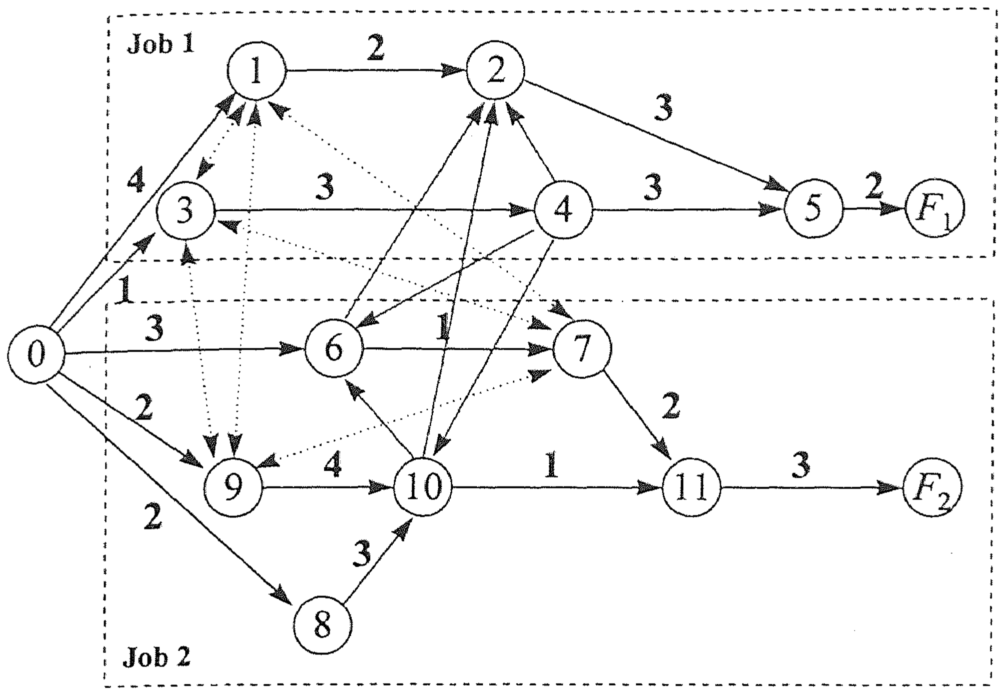


Figure 3-4: Complete shop scheduling graph with partial sequence

Graph Decomposition Algorithm

The following algorithm decomposes the scheduling graph to a sub-problem that associates with workstation *k*. The objective of the new sub-problem is similar to the one in the original problem.

```
for($n$ = 1 to (# of nodes in the graph))
    if ( (node $n$ is not assigned to the workstation $k$) and
         (node $n$ is not the source or the sink) )
        ⇒ Create arcs connecting the prior nodes with the post nodes with weight
           equals the weight of prior arc + the weight of post arc.
        ⇒ Remove node $n$ and arcs connecting to that node.
```

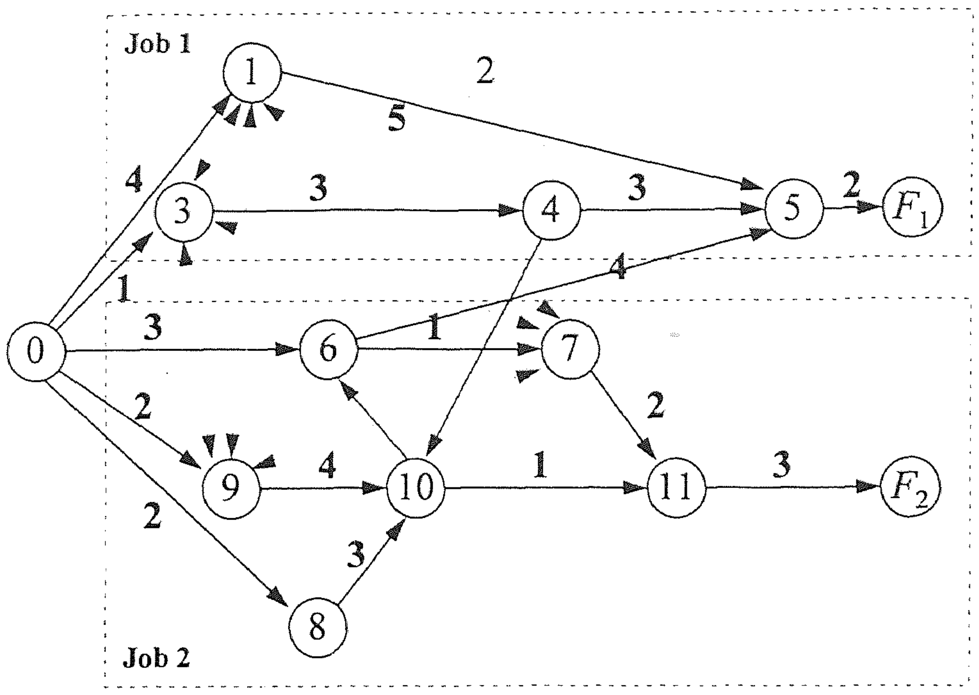


Figure 3-5: Removing node 2

The following example shows the steps on decomposing the graph into the sub-problem that contains machine 1. First, node 2 is selected. There are 3 arcs connecting to this node – (1,2), (6,2) and (2,5) with weight 2, 1 and 3 respectively. Arc(1,5) and (6,5) are added with weight 5 and 4. After node 2 is removed, the new graph is shown in Figure 3-5.

Figure 3-6 shows the final result from the graph decomposition algorithm. This graph shows that there are process dependency between operations 3 and 7, and operations 9 and 7. The minimum completion time of jobs 1 and 2 is 14. Any sequence applied to graph in Figure 3-6 will provide the same completion time as of the one applied to the original graph (Figure 3-4).

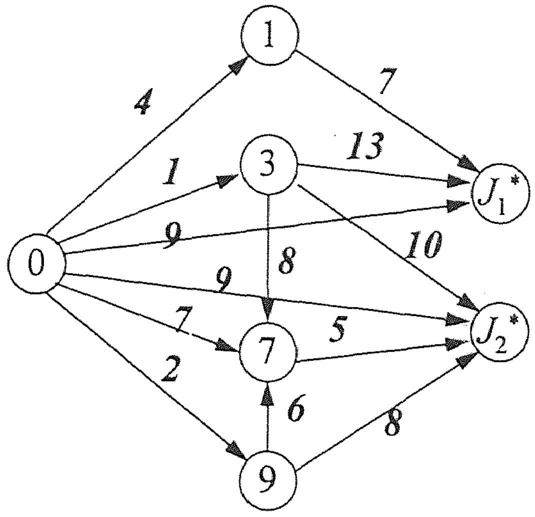


Figure 3-6: Final decomposed graph

3.3 Sub-Problems

After a decomposed graph is derived, we can formulate the sub-problem. The problem will be fairly similar to the original one; however, only a single workstation will be concerned, say workstation i. The graph decomposition method removes the operations that are not related to workstation i and replaces the arc by the combined processing time. On the decomposed problem, the number of jobs, jobs' due dates and their priorities remain the same as in the original problem. The number of tasks to be sequenced is equal to the number of operations to be processed on workstation i. The release times of these operations are modified. The new release times are determined from EST of the decomposed graph. After these operations are completed, tail operations must be performed. The job is considered completed when all its prior tails are finished. The processing time of the tail determines from its length less the processing time of the prior operation. The objective of the problem is to find the sequence for the workstation to minimize the weighted tardiness. The sub-problem can be a single machine, parallel machines or batch machine workstation depending on the structure of the shop. In the following section, we describe each sub-problem. They are discussed further in Chapters 4, 6, and 7.

3.3.1 Single Machine Scheduling with Tails

There are n jobs to be processed. Job j has due date, $d^{(j)}$, penalty for being late, $w^{(j)}$, and the penalty for being early, $-h^{(j)}$. They need to be processed on machines according to their routing. Let us define an *operation* as a processing on a particular machine. The jobs can be perceived as a network of operations. In this problem, the processing of each job is divided into two stages. There are m operations that have to be processed on a single machine in first stage. Some of these operations may have precedent relations with others. Let $Q = \{ (i,i'); i$ and i' are operations that have precedent relation$\}$ represent the set of delayed precedent constraints. The releasing time of operation i is r_i and the processing time is p_i. The starting time of operation i can be expressed as $t_i \geq \max(r_i, t_{i'} + p_{ii'})$ where $p_{ii'}$ is the delayed time constraint between operation i and i'. Job preemption is not allowed.

After operation i on the first stage is completed, its succeeding operations (tails), $i^{(1)}$, $i^{(2)}$, $i^{(3)}$, ..., $i^{(n)}$, on the second stage can be started. There are no machine constraints for these operations. The operations will start promptly upon the completion of operation i. Let $q_i'^{(j)}$ represent the processing time of operation $i^{(j)}$ and $q_i^{(j)} = p_i + q_i'^{(j)}$. The completion time of job j, $C^{(j)}$, is determined from the completion time of operation $1^{(j)}$, $2^{(j)}$, $3^{(j)}$,..., $m^{(j)}$ or $C^{(j)} = \max(rc^{(j)}, \max_i(t_i + q_i^{(j)}))$ where $rc^{(j)}$ is the minimum completion time of job j and t_i is the starting time of operation i. The processing diagram is shown in Figure 3-7.

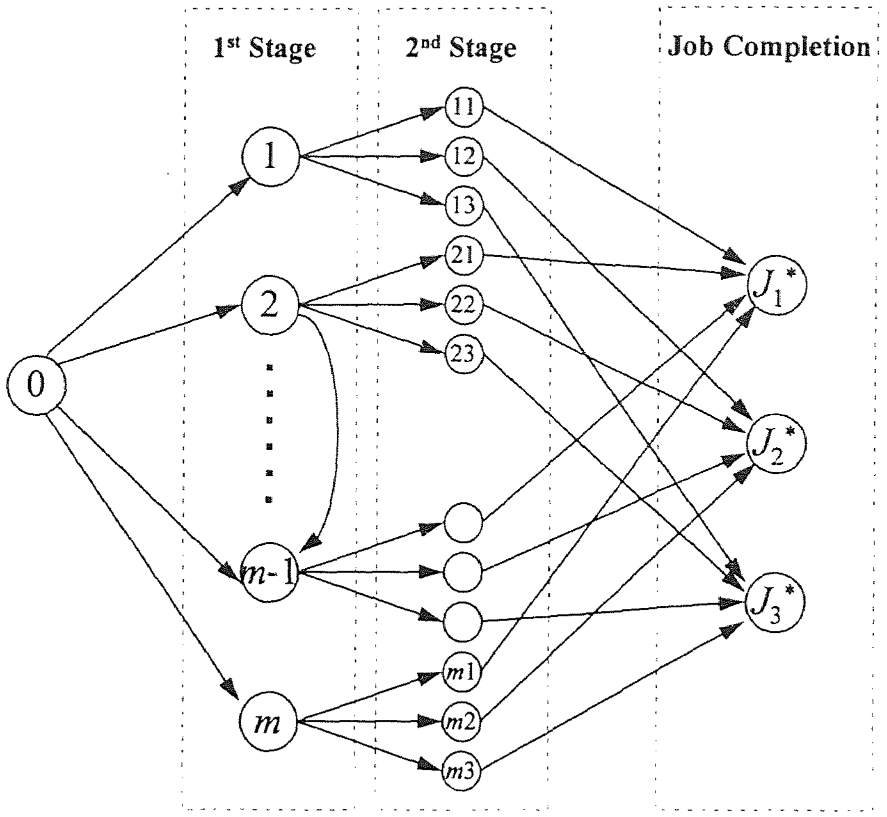


Figure 3-7: A shop with 3 jobs and m operations on stage 1

The penalty for job j is $w^{(j)}T^{(j)}$ if it is late and $-h^{(j)}E^{(j)}$ if it is early where T is the job tardiness, and E is the job earliness. In the other words, there is reward for completing the job early. We are interested in finding a sequence of the machine in the first stage that minimizes the total penalty, $\min\sum_{j=1}^{n}\left\{w^{(j)}T^{(j)}-h^{(j)}E^{(j)}\right\}$. We are interested on the case that $h^{(j)}=\frac{k}{w^{(j)}}$ and $w^{(j)} >> h^{(j)}$ or $0 < k << 1$. The reason behind this is discussed in section 7.2.2. It is important to restrict $w^{(j)}$ to be greater than zero. This objective can be considered as a generalization of weighted tardiness objective.

A sequencing graph for job j is shown in Figure 3-8. The release time of operation i is r_i. There is an arc connecting operation 2 and operation m-1 representing the delayed precedent constraint. Operation m-1 cannot start before time $t_2+p_{2(m-1)}$. Figure 3-9 shows the complete graph where all jobs are shown.

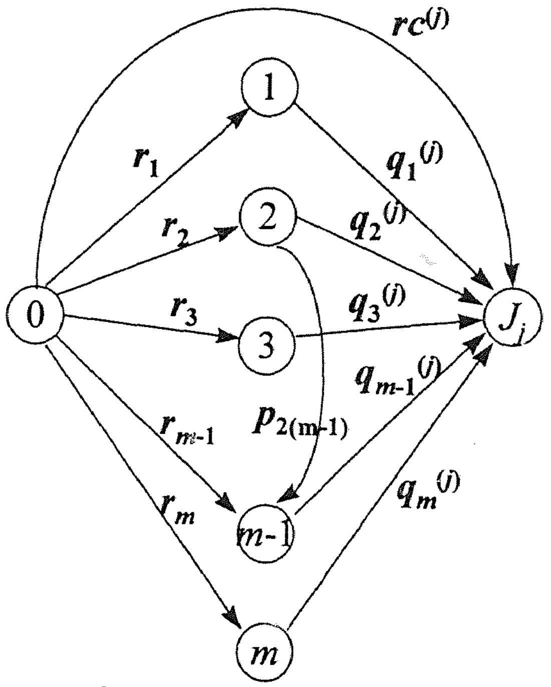


Figure 3-8: Sequencing graph for job j

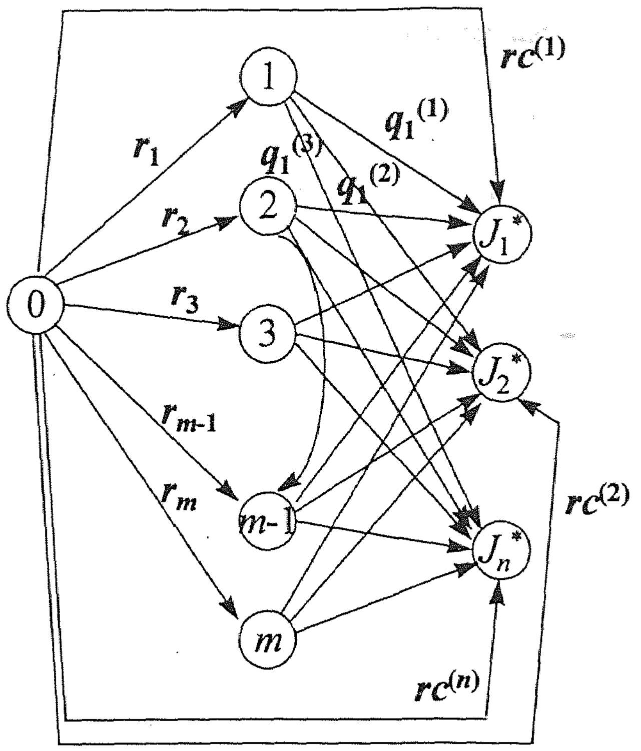


Figure 3-9: Complete sequencing graph

3.3.2 Parallel Machine Scheduling with Tails

The following is a scheduling problem for a c-identical parallel machines workstation. There are m operations to be completed. The operation can be processed on any machine in the workstation. The machine can process only one operation at a particular time and no preemption is allowed. The processing time and release time of operation i is p_i and r_i. Some of these operations may have precedent relations with others. Let $Q = \{ (i,i'); i$ and i' are operations that has precedent relation$\}$ represent the set of delayed precedent constraints. The starting time of operation i can be expressed as $t_j \geq \max(r_j, t_i + p_{ii'})$ where $p_{ii'}$ is the delayed time constraint between operation i and i'.

After operation i is completed, its tails, $i^{(1)}, i^{(2)}, .., i^{(n)}$, will be started immediately. There is no machine constraints for these tails. The processing time of tail $i^{(j)}$ is $q_i^{(j)}$

where i indicates the preceding operation and j indicates the related job. Among n jobs, job j is considered completed when all of its preceding operations and their tails that leads to job j, $1^{(j)}$, $2^{(j)}$, $3^{(j)}$, .., $m^{(j)}$, are completed. Job j has due date $d^{(j)}$, penalty for being late for one unit time $w^{(j)} > 0$, and penalty for being early $-h^{(j)}$. The completion time of job j, $C^{(j)}$, is determined from the completion time of operation $1^{(j)}$, $2^{(j)}$, $3^{(j)}$, ..., $m^{(j)}$ or $C^{(j)} = \max(rc^{(j)}, \max_i(t_i + q_i^{(j)}))$ where $rc^{(j)}$ is the minimum completion time of job j and t_i is the starting time of operation i. The penalty for job j is $w^{(j)}T^{(j)}$ if it is late and $-h^{(j)}E^{(j)}$ if it is early. The objective is to find the schedule for all the machines in the workstation to minimize the total penalty, $\min \sum_{j=1}^{n} \{w^{(j)}T^{(j)} - h^{(j)}E^{(j)}\}$. We are interested on the case that $h^{(j)} = \frac{k}{w^{(j)}}$ and $w^{(j)} >> h^{(j)}$ or $0 < k << 1$ (refer to section 7.2.2).

3.3.3 Batch Machine Scheduling with Tails

There are m tasks needed to be scheduled on a machine. This machine can process up to z tasks in a single run. After the processing has started, it cannot be interrupted until it completes. The release time of task i is r_i. There are delayed precedent constraints among some tasks. Let $Q = \{ (i,i'); i \text{ and } i' \text{ are tasks that has precedent relation}\}$ represents the set of delayed precedent constraints. The starting time of task i can be expressed as $t_j \geq \max(r_j, t_i + p_{ii'})$ where $p_{ii'}$ is the delayed time constraint between task i and j. After task i has been completed, its tail processing, $i^{(1)}$, $i^{(2)}$, ..., $i^{(n)}$, will be started immediately. There is no machine constraints for these tails. The processing time of tail $i^{(j)}$ is $q_i^{(j)}$ where i indicates the preceding task and j indicates the related job. Job j is considered complete when all of its preceding tasks and their tails that pointed to, $q_i^{(j)}$ for all i, are completed. There is a due date, $d^{(j)}$, tardiness penalty, $w^{(j)}$, earliness penalty, $-h^{(j)}$ and minimum completion time, $rc^{(j)}$, for job j.

The batch machine can start processing when there is at least one task waiting. It has a capability to process z tasks in a single run; however, it does not need to wait until all z tasks have arrived to begin the processing. The processing time of the batch is p. The machine can process only one batch in a single run. We assume that there is no setup time between batches.

The objective of the problem is to minimize the penalty (weighted tardiness and earliness). We do not consider the cost of production. Therefore, the number of runs is not in our consideration. The decision variables are the tasks assignment and the batch starting time.

Example 3-1: A simple assembly shop problem is constructed as follows. There are three jobs to be processed on three workstations. The information on workstations and jobs are provided in the Table 3-2 to Table 3-4. The routing of the jobs are shown in Figure 3-10. In the following, we show three decomposed sub-problems derived from the original problem.

Table 3-2: Workstations in Example 3-1

Workstation	Type	Description
1	parallel machines (2 m/c)	Cutting
2	batch machine (size 2)	Pressing
3	single machine	Assembling

Table 3-3: Jobs in Example 3-1

Job	Release time	Due date	Priority
1	0	19	1
2	1	14	2
3	2	25	1

Table 3-4: Jobs and operations in Example 3-1

Job	Operation	Workstation	Preceding operations	Processing time
1	1	1	-	3
	2	1	-	4
	3	3	1, 2	6
	4	2	-	2
	5	3	3, 4	6
2	6	1	-	5
	7	2	-	5
	8	1	-	3
	9	3	6, 7, 8	4
	10	2	9	2
3	11	1	-	6
	12	2	11	2
	13	1	-	4
	14	3	12, 13	5
	15	2	14	3

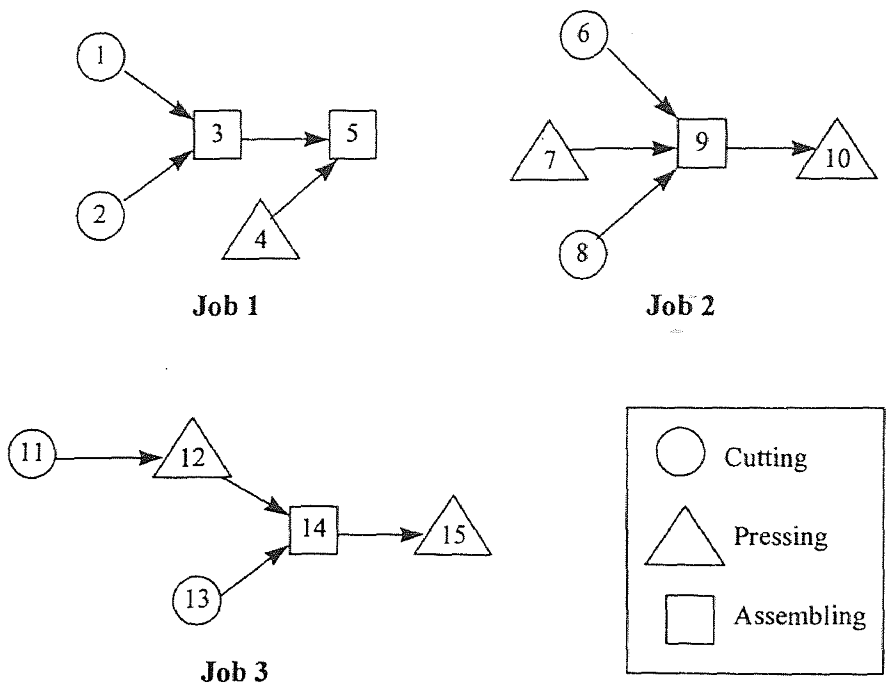


Figure 3-10: Jobs routing

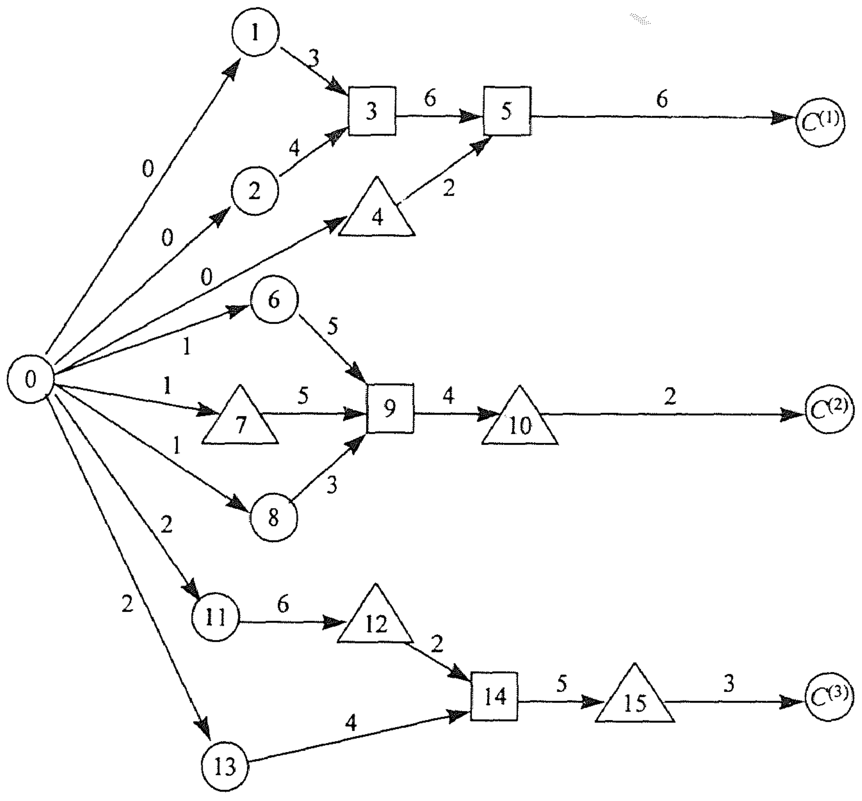


Figure 3-11: Directed graph

The directed graph can be created by adding the source and sinks to the job routes as shown in Figure 3-11. Figure 3-12 to Figure 3-14 show the decomposed graphs for workstations 1 to 3 respectively.

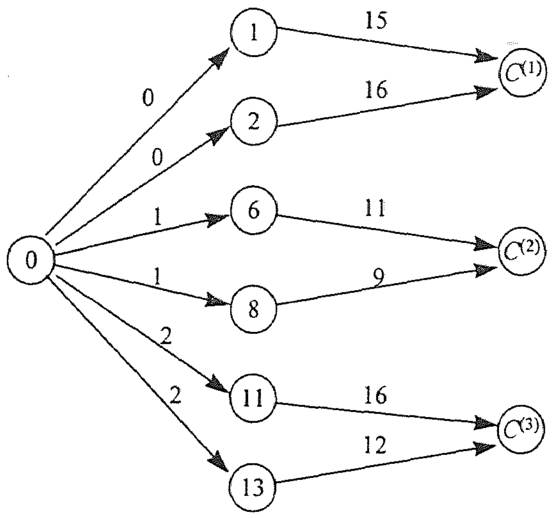


Figure 3-12: Decomposed graph on workstation 1

The decomposed problem for cutting workstation (sub-problem 1) is as follows. There are six tasks to be scheduled on two identical cutters. Jobs can be processed on any of these two machines. Each task has the release time, processing time, and tails as shown in the following table. Job j is considered completed when tail $q_i'^{(j)}$ for all i are finished. The objective of the problem is to find the sequence on the cutting workstation that minimizes $\sum_{j=1}^{3} w^{(j)} T^{(j)}$. Table 3-5 and Table 3-6

Table 3-5: Operation information on sub-problem 1

Task (i)	Release time (r_i)	Processing time (p_i)	Tail (1), $q_i'^{(1)}$	Tail (2), $q_i'^{(2)}$	Tail (3), $q_i'^{(3)}$
1	0	3	12	-	-
2	0	4	12	-	-
6	1	5	-	6	-
8	1	3	-	6	-
11	2	6	-	-	10
13	2	4	-	-	8

Table 3-6: Job information on sub-problem 1

Job	Due date	Priority
1	19	1
2	14	2
3	25	1

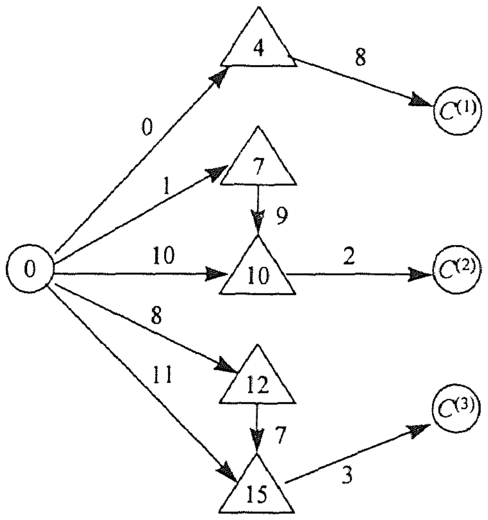


Figure 3-13: Decomposed graph for workstation 2

The second sub-problem decomposed from pressing workstation is fairly similar to the previous one. However, there are two delay precedent constraints, $p_{7,10} = 9$ and $p_{12,15} = 7$ and the tasks are processed in batches. The press machine can work on two tasks in the single run. It may start immediately when a task arrives or may wait until the second task comes. The data is provided in Table 3-7 and Table 3-8.

Table 3-7: Operation information on sub-problem 2

Task (i)	Release time (r_i)	Processing time (p_i)	Tail (1), $q_i'^{(1)}$	Tail (2), $q_i'^{(2)}$	Tail (3), $q_i'^{(3)}$
4	0	2	6	-	-
7	1	2	-	-	-
10	10	2	-	0	-
12	8	2	-	-	-
15	11	2	-	-	1

Table 3-8: Job information on sub-problem 2

Job	Due date	Priority
1	19	1
2	14	2
3	25	1

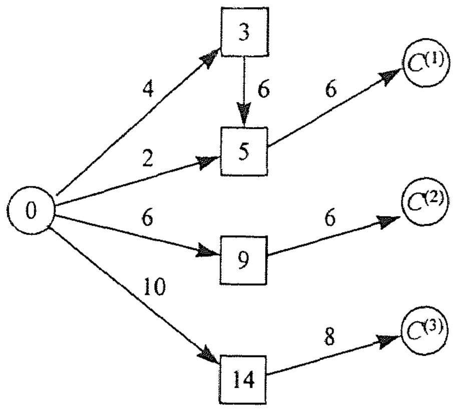


Figure 3-14: Decomposed graph on workstation 3

The assembly workstation is a single machine workstation. There are four tasks to be processed on this machine. There is a delay precedent constraint, $p_{3,5} = 6$, on the third sub-problem (assembling). The data on the problem is provided in Table 3-9 and Table 3-10.

Table 3-9: Operation information on sub-problem 3

Task (i)	Release time (r_i)	Processing time (p_i)	Tail (1), $q_i'^{(1)}$	Tail (2), $q_i'^{(2)}$	Tail (3), $q_i'^{(3)}$
3	4	3	-	-	-
5	2	3	3	-	-
9	6	3	-	3	-
14	10	3	-	-	5

Table 3-10: Job information for sub-problem 3

Job	Due date	Priority
1	19	1
2	14	2
3	25	1

3.4 Heuristic Development

The next three chapters are devoted to sub-problems for single machine, parallel machine and batch machine workstations respectively. Each of the problem is different from general scheduling problems as tail processing is included. We develop a number of techniques to solve these sub-problems. The results are compared to some standard priority rules as the optimal solutions are prohibitive because of the problem nature (NP-completed). These sub-problems constitute the key components for the assembly shop scheduling discussed in later chapters.

In chapter 7, we discuss the aggregated problem. This chapter requires the results from the previous three chapters. We first study the assembly shop problem in single machine workstations environment, and we, then, extend the model to include all the three types of workstations.

Chapter 8 explains an extension of the assembly job shop scheduling to other objectives including minimizing the maximum completion time, minimizing the weighted flow time, and minimizing the maximum weighted lateness, etc. The results reported are promising. We attached all the developed heuristics to LEKIN, an assembly shop scheduling system. The system provides a linkage between the theoretical research and the practitioners.

CHAPTER 4

SINGLE MACHINE SCHEDULING PROBLEM WITH TAILS (SMSPT)

The single machine scheduling problem discussed in this chapter is formulated differently from general single machine scheduling problems. This problem stems from the decomposition technique in chapter 3. After the jobs have been completed, there is a number of tail operations that follow. The completion times of tails determines the objective value which is to minimize the weighted tardiness & earliness (see chapter 3 for details). In this chapter, we propose a priority rule and a heuristic based on critical path analysis. When dependent setup time is a concern, we extend the heuristic by including the local searches. The results in this chapter are essential for constructing the assembly shop scheduling.

4.1 Mathematical Formulation

The objective of the problem is to find the sequence on a single machine that minimizes the weighted tardiness and earliness. The processing can be divided into two stages. After the processing on the operation (on the first stage) is completed, the tail processing (the second stage) can start. The job is considered completed when all its prior tails have finished. Each job has its own due date and weight. Details on the problem description can be found in chapter 3.

$$\text{Min} \sum_{j=1}^{n} (w^{(j)} T^{(j)} - h^{(j)} E^{(j)})$$

subject to

(1) $t_i \geq r_i,$ $i = 1, \ldots, m,$

(2) $C^{(j)} \geq t_i + p_i + q_i'^{(j)},$ $j = 1, \ldots, n; i = 1, \ldots, m,$

(3) $C^{(j)} \geq rc^{(j)},$ $j = 1, \ldots, n,$

(4) $t_i \geq t_{i'} + p_{i'}$ or $t_{i'} \geq t_i + p_i,$ $i, i' \in S,$

(5) $t_{i'} \geq t_i + p_{ii'},$ $(i, i') \in Q,$

(6) $L^{(j)} = C^{(j)} - d^{(j)},$ $j = 1, \ldots, n,$

(7) $T^{(j)} \geq 0$ and $T^{(j)} \geq L^{(j)},$ $j = 1, \ldots, n,$

(8) $E^{(j)} \geq 0$ and $E^{(j)} \geq -L^{(j)},$ $j = 1, \ldots, n,$

where,

t_i is the starting time of operation i,

$C^{(j)}$ is the completion time of job j,

p_i is the processing time of operation i,

S is the set of operations,

Q is the set of delayed precedent constraints,

$q_i'^{(j)}$ is the tail processing time toward job j and following operation i,

rc$^{(j)}$ is the lower limit of the completion time for job j,

$p_{ii'}$ is the delayed time between operation i and i',

$L^{(j)}$ is the lateness of job j,

$T^{(j)}$ is the tardiness of job j,

$E^{(j)}$ is the earliness of job j.

Constraint Interpretation

(1) Release time constraints

(2) Jobs completion time determination

(3) Lower limit of the completion time

(4) Machine capacity constraints: machine can process single operation at a time.

(5) Delayed precedent constraints

(6) Determine the job lateness

(7) Determine the job tardiness

(8) Determine the job earliness

The constraint (4) is the disjunctive constraint. It states that operation i may be processed either before operation i' or after but they cannot be processed at the same time. The problem can be solved using disjunctive programming. The original problem will be divided into sub-problems. Only one inequality is selected for each instance. After the sub-problems are solved, the final solution can be determined iteratively. When the number of sub-problems is large, it may not be computationally economical to find the optimal solution.

The disjunctive programming problem can be formulated as a 0-1 integer programming by modifying constraint (4). The new 0-1 variable, $x_{ii'}$, is 1 if i is scheduled before i' and 0 otherwise. The following is the formulation of the new constraints that replaces (4).

(9) $t_i \geq t_{i'} + p_{i'} - BigM\, x_{ii'}$ $\quad , i, i' \in S,$

(10) $t_{i'} \geq t_i + p_i - BigM(1\text{-}x_{ii'})$ $\quad , i, i' \in S,$

where, *BigM* is a large number.

As the problem is one of NP-hard problem, there is no polynomial time algorithm to solve it optimally. The following section discusses some properties that will be used to develop the heuristic.

4.2 Scheduling Graph

The disjunctive graph technique developed by Balas (1969) is modified and applied to the problem. The graph has m nodes in the first layer and n nodes in the next layer. The nodes in the first layer represent the operations while the nodes in the second layer represent job completions. The conjunctive arcs are used to represent the operation precedent constraints. Their weights are assigned according to the delayed time. The source, node 0, is added. There are m arcs from source to each operation on the first layer with weight r_i. Similarly, there are n arcs from source to each operation in the second layer with weight $rc^{(j)}$. The delayed precedent constraints are added to the graph by inserting conjunctive arcs with weight $p_{ii'}$ linking operation i and i'. The disjunctive arc pairs (two conjunctive arcs in opposite direction) representing disjunctive constraints are added between operations in the first layer that do not have delayed precedent constraint. The arc from node i has weight p_i.

This graphical representation contains the information on job releasing time, precedent relationship and processing time. Figure 4-1 shows the disjunctive graph representation. The precedent relations (conjunctive arcs) are represented by dashed lines. The disjunctive arcs pairs are shown in solid lines. Sequencing the machine is similar to selecting an arc from each disjunctive arc pair without creating a cycle. These selected arcs are called a *selection* (Figure 4-2 and Figure 4-3). The selection can be reduced to Figure 4-5 for a better interpretation.

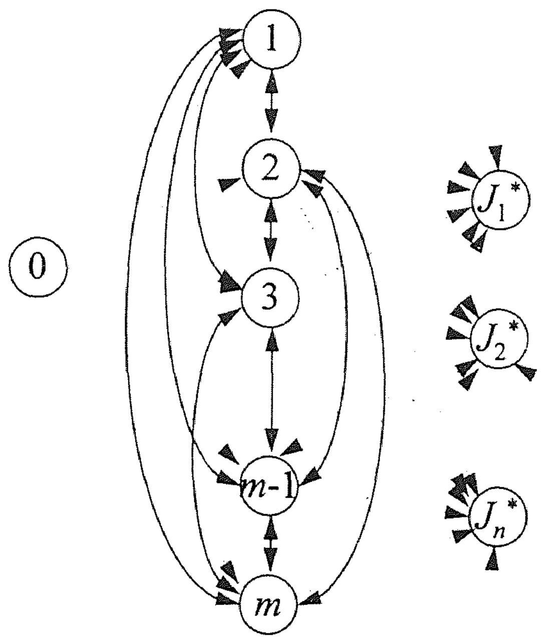


Figure 4-1: Disjunctive graph representation

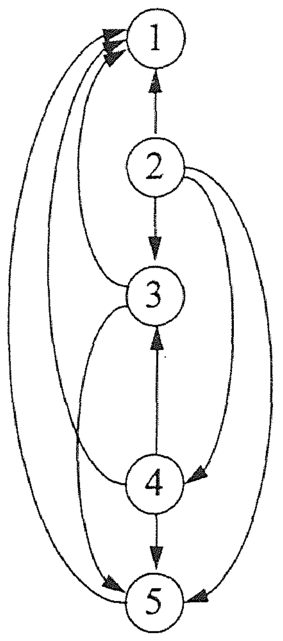


Figure 4-2: A feasible selection [2-4-3-5-1]

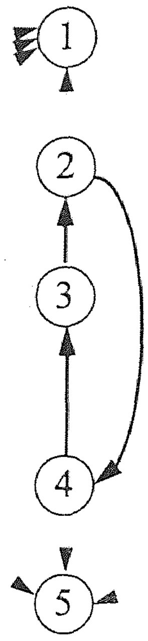


Figure 4-3: A cycle in the graph

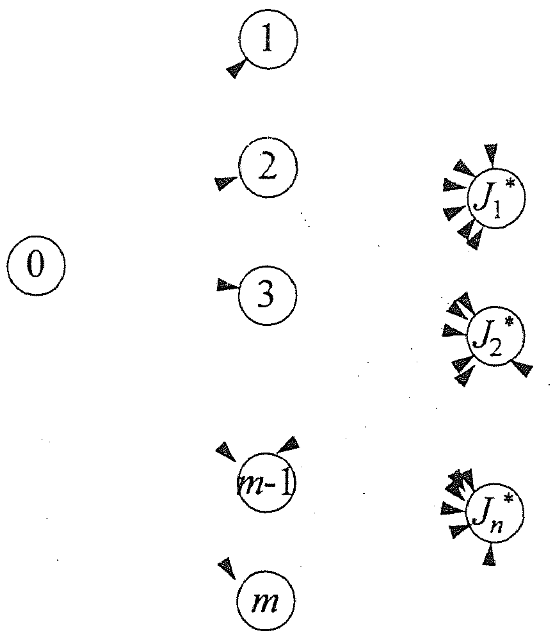


Figure 4-4: Directed graph

When the disjunctive arc pairs are removed from the graph, the rest of the graph is called directed graph (Figure 4-4). It represents the relaxation of the problem as the disjunctive constraints are removed. Therefore, it can be used to provide the lower bound of the job completion time.

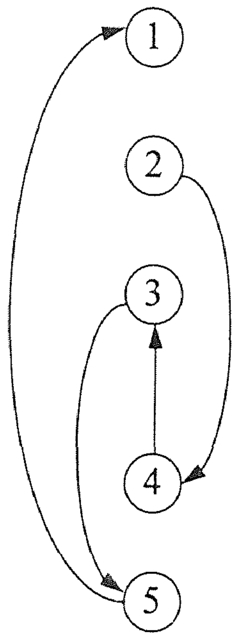


Figure 4-5: The reduced selection

4.3 TER (Tardiness - Earliness Rule)

The structure of this problem is unique. The completion time of each job depends on the starting time and processing time of many operations. Job weights cannot be assigned directly to the operation. The commonly used indexing rules that incorporate the job weights in the indexing function such as WSPT (Weighted Shortest Processing Time), ATC (Apparent Tardiness Cost), etc. cannot be applied directly to the problem.

We propose a new indexing rule specially designed for this problem. The extension of this rule will cover the general case where operation release times are different and the operation dependencies are considered. This rule determines the sequence dynamically. On any iteration, the operation indices are calculated. Starting from time 0, it finds an operation with the highest index. Once an operation is sequenced, time t is updated. The next operation to be sequenced is determined. The process continues until all operations are sequenced.

Index Function

As our objective value depends on the job completion times and job weights, high priority should be given to the operations that have already delayed some jobs. Delaying these tasks will definitely increase the objective value. Among these operations, the one that has strong effects on the completion time of multiple jobs should be provided with high priority. We may judge them by using the weighted tardiness value.

When there is no operation that has already been delayed, the high priority should be assigned to the operation that has less slacks. In other words, it is the operation that pushes the completion time of jobs furthest. To merge these two properties, we use exponential function to decrease the intensity of the earliness factor. Then, combine the tardiness and earliness factors into a function. The function can be written as

$$I_i(t) = \sum_j w_j (T_j + e^{E_j}).$$

Figure 4-6 shows the characteristic curve of the function.

This index function is a combination of exponential and linear function. The task that is linked to early jobs will be assigned with lower priority comparing to the one linked to tardy jobs. We avoid the use of parameters that need to be determined by statistical method in the function, e.g. ones in ATCS rule. Although these parameters may provide a better schedule, we need to analyze historical data of the machine shop to determine their values.

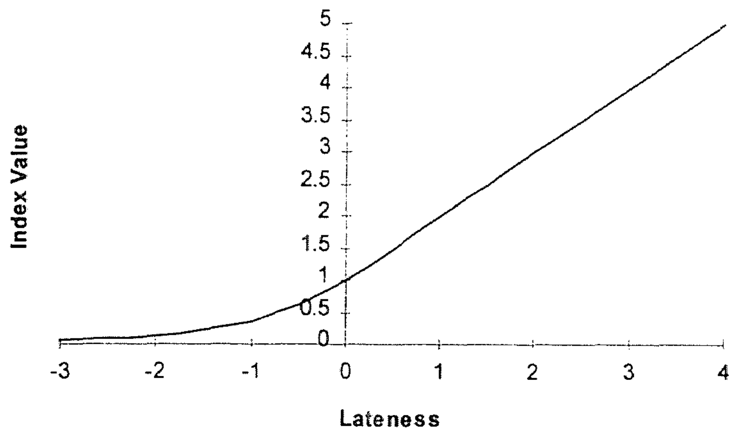


Figure 4-6: Characteristic curve to the index

The Procedure

The simplest TER is developed for the case that all tasks are available at time zero (r_i = 0) and there is no delay precedent constraint. The heuristic is repeated iteratively. At each step, the rule identifies the operation that has the longest impact on delaying the jobs. It is the operation that provides the highest index value. The steps are as follows.

(i) Let U be the set of unscheduled operations, $S = \varnothing$ be the set of scheduled tasks, operation starting time, $t_i = 0$ for $i = 1, \ldots, m$ and $t = 0$.

(ii) Determine $I_j(t)$ for operation $j \in U$.

(a) Let $t'_j = t_j$ and $t_j = t$.

(b) Let $I_j(t) = \sum_k w^{(k)}(T^{(k)} + e^{E^{(k)}})$,

where $T^{(k)} = \max(0, C^{(k)} - d^{(k)})$,

$E^{(k)} = \max(0, d^{(k)} - C^{(k)})$,

$C^{(k)} = \max_{i=1,\dots,m}\{t_i + q_i^{(k)}\}$.

(c) Let $t_j = t'_j$.

(iii) Select k; $I_k(t) \geq I_j(t)$ for $j \in U$.

(iv) Let $U = U - \{k\}$, $S = S + \{k\}$, $t_k = t$, and $t = t + p_k$.

(v) Repeat (ii)-(iv) until $U = \varnothing$.

4.4 Dynamic Job Arrivals

The priority rule, TER, that has been developed in the previous section does not consider dynamic job arrivals and job precedent constraints. To solve this new problem, we apply the concept based on the algorithm by Schrage. In his algorithm, a set of ready jobs is identified. The job with the highest index (remaining processing time) in the set is selected as the next job to be processed. Though his algorithm was not designed for job dependency, it could be included without any difficulty.

TER is a one pass heuristic. It selects the operation greedily based on an index function. Though it cannot guarantee to provide the optimal sequence, it gives a reasonably good and feasible one with a very short computer running time. Starting at time 0, the ready operations are identified and placed into a *ready set*. These operations are released before time t. In order for an operation to become ready, it should not have any prior precedent operations. If it has, those operations should have been sequenced. Consequently, in order for operation j to be ready, the following equation must be complied.

$$t \geq \max(r_i, t_i + p_{ii'}) \quad \text{for } \forall (i,i') \in Q.$$

In the case that the set is empty, time t will be advanced to the next operation arrival time, $t = \min_{i'} \max(r_{i'}, t_i + p_{ii'})$. The ready set is kept with some members at any moment. The next operation to be sequenced is determined from the operations in the ready set. The operation that has the highest index is selected and removed from the set. The steps are repeated until all operations are sequenced.

In TER, U is a set of unscheduled operations, A is a set of ready operations (no precedent constraint), and t is the current sequencing time. The steps are as follows.

(i) Let U be a set of unscheduled jobs; $A = \varnothing$; $t = 0$; G = directed graph; $l = 0$;

$rl_k = r_k$ for $k = 1,\ldots,m$.

(ii) $A = A \cup \{j, rl_j \leq t; j \in U\}$; $U = U - \{j\}$. If $A = \varnothing$ and $U = \varnothing$, stop.

Else, if $A = \varnothing$ and $U \neq \varnothing$, $A = A \cup \{j, rl_j = \min_{k \in U} rl_k; j \in U\}$; $U = U - \{j\}$.

(iii) Schedule $\{j\}$ such that $j \in A$ and $I_j(t) \leq I_k(t)$; $k \in A$.

$A = A - \{j\}$. $t_j = \max\{rl_j, t\}$. $t = \max\{rl_j, t\} + p_j$. $G = G \cup (l,j)$.

$rl_k = t + p_{jk}$ if $(j,k) \in Q$. $l = j$.

(iv) Goto (ii).

Determine the Directed Graph

The directed graph $D(N,A)$ is obtained from the problem. Node set N corresponds to the set of operations. One source node and n sink nodes are added. The sinks represent the jobs completion. The precedent relationships are represented by conjunctive arc set A. There are conjunctive arcs linking source to node i with weight r_i, node i to sink j with weight $q_i^{(j)}$ and source to sink j with weight $rc^{(j)}$. If delayed precedent constraint between operation i and j exists, add the arc (i, i') with arc length $p_{ii'}$ to A.

Determine the Index

Given graph G which is the directed graph D with partial selection (starting from no selection), the index for operation i' is determined by adding the conjunctive arc (i,i') where i is the previously sequenced operation to G. The critical path lengths can be determined from CPM method. Each critical path determines the completion time of the associated job when relaxing machine constraints for the operations that have not been sequenced. The index value is determined from the index function

$$I_i = \sum_j w^{(j)}(T^{(j)} + e^{E^{(j)}}),$$

where $T^{(j)} = \max(0, C^{(j)} - d^{(j)})$,

$E^{(j)} = \max(0, d^{(j)} - C^{(j)})$,

$C^{(j)}$ is the length of the critical path associated to node k or the EST of sink j.

4.5 Critical Arcs and Cluster

Critical arcs are defined as the arcs which pass through *critical operations* forming the critical path. If any of the critical operations is delayed, it will directly effect the lengths of some critical paths which are equivalent to the completion time of some jobs. *Critical tail* is the tail of a critical operation that is on the critical path. That critical operation is called *critical tail operation.* In Figure 4-7, there are three critical paths: 0-1-J_1^*, 0-1-2-3-J_2^*, and 0-4-5- J_n^*. Operations 1 to 5 are critical operations as they are on the critical paths. There are three critical tails -- (1- J_1^*), (3, J_2^*) and (5, J_n^*), one corresponding to each critical path. Operations 1, 3, and 5 are critical tail operations.

Critical arcs can be determined by applying CPM technique in project management. First, let $t_0 = 0$. The earliest starting time (EST) of nodes can be determined from the release time and the starting time of their prior nodes. After EST of sinks are determined which are the job completion times, the latest starting time (LST) can be determined. See Section 2.6.1 for further details. The critical operation is the operation that has EST = LST. Delaying these operations will result in the extension of completion time for some jobs. There will be at least m sets of critical arcs in the graph.

Cluster is a set of adjacent operations in the sequence that has no idle time in between (Figure 4-8). According to TER/D, a new cluster is defined when $A = \varnothing$ and $U \neq \varnothing$ on step (ii). The machine idle time occurs only when there is no job available to sequence at that moment. Figure 4-7 shows an example of critical arcs and clusters in the graph.

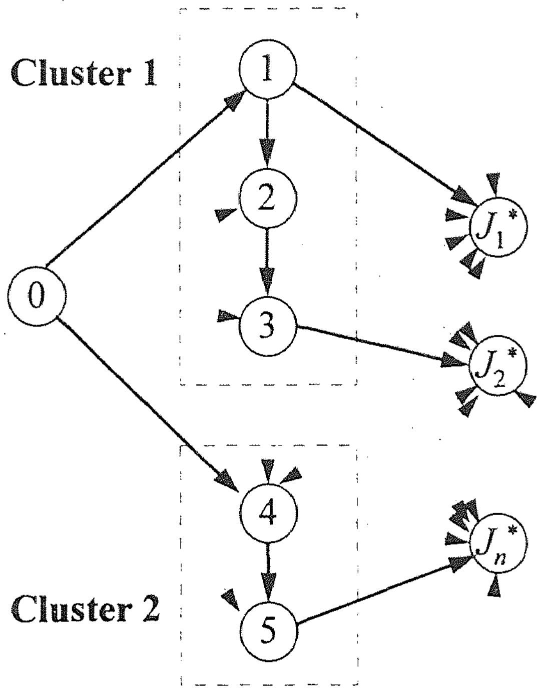


Figure 4-7: Critical arcs and clusters

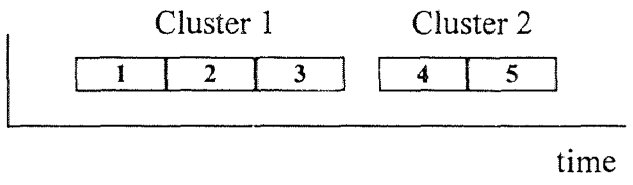


Figure 4-8: Gantt chart representing the two clusters

4.6 Critical Path Analysis

Proposition 1: The critical arc always passes through the first operation in the cluster.

Proof: From the definition of cluster and the TER.

Proposition 2: Moving operation i from cluster c to cluster c+1 may provide benefit only when

1) $r_{i+1} > t_i$

2) The idle time between cluster c and c+1 is less than $p_i - t_{i+1} + \max(r_{i+1}, t_i)$.

Proof: If the idle time between cluster c and c+1 is greater than $p_i - t_{i+1} + \max(r_{i+1}, t_i)$, then switching the operation will not merge the two clusters. There always has a better sequence that operation i is sequence last in cluster c.

4.7 Cluster Sequencing

Since the problem is NP-hard, we will not try to explore the optimal solution. A heuristic to find a near optimal sequence is preferred. The method starts with TER. Operations are separated into clusters. Then, we try to improve the sequence in each cluster by moving operations in the cluster. With critical path analysis, we can trim a large number of possibilities in moving down tremendously. The following properties provide some conditions for the moves. Let us define *move(i,j)* as the sequence after removing operation i from the sequence and insert it back right after operation j when i comes before j and *moveback(j,i)* as removing operation j from the sequence and insert it back right before i.

Lemma 1: Re-sequencing the cluster that its sequence does not contain critical arc will not provide improvement.

Proof: Current sequence has no direct effect to the completion time of any job. Therefore, a better sequence for that cluster cannot be found.

<u>One Critical Arc in the Cluster</u>

Lemma 2: If the critical arc is on the first operation in the cluster, the sequence of that cluster is optimal.

Proof: The completion time of the job cannot be reduced any further. Therefore, the current sequence is optimal.

Lemma 3: The lower bound of the job completion time is $r_i + \sum_{j=i}^{l} p_j + \min_{j=i..l} q'^{(k)}_j$, where i is the first operation in the cluster, l is the last operation in the cluster, and k is the job that is on the critical arc. Hence, if the critical arc passes through $\min_{j=i..l}\left(q'^{(k)}_j\right)$ arc, the sequence is optimal.

<u>Many Critical Arcs in the Cluster</u>

Proposition 3: Moving operations that follow the last critical tail operation in a cluster will not improve the objective value.

Proof: The objective value is determined by the lengths of the critical paths. Moving the operations that follow the last critical operation will not shorten the critical paths. Therefore, it cannot reduce the completion time of any job.

Proposition 4: *Move*(i,j) when i and j are operations between two adjacent critical tail operations or the first critical operation and the following critical tail operation in the same cluster and i precedes j will not improve the objective value.

Proof: Let $S = \{\ldots, n_1\ldots, i, \ldots, j, \ldots, n_2, \ldots\}$ where operation n_1 is the first critical operation or a critical tail operation in cluster l and n_2 is the following critical tail operation linked to job k in the same cluster. Figure 4-9 shows the move according to Proposition 4 which does not improve the objective value. The completion time of job k before the move is $L(i,j) = r_{n_1} + p_\alpha + p_i + p_\beta + p_j + p_\chi + q_{n_2}^{(k)}$ where $p_\alpha = \sum_{g=n_1}^{i-1} p_g$, $p_\beta = \sum_{g=i+1}^{j-1} p_g$, and $p_\chi = \sum_{g=j+1}^{n_2-1} p_g$. After the move, the completion time of job k will be $L(j,i) \geq \max\left(r_{n_1} + p_\alpha, r_{i+1}\right) + p_\beta + p_j + p_i + p_\chi + q_{n_2}^{(k)}$ while the completion time of other jobs does not decrease. As $L(j,i) \geq L(i,j)$, the property follows. In the same fashion, *move*(j,i) will not decrease the objective value.

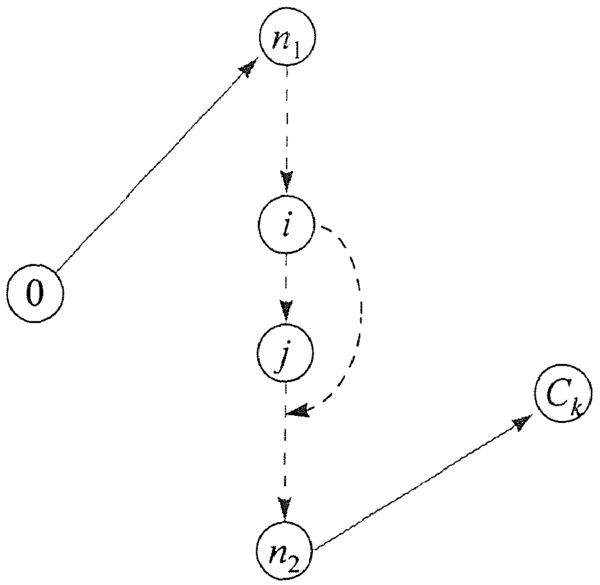


Figure 4-9: Move(i,j) between the first critical operation and the critical tail operation

Lemma 4: Move(i, j) where j is a critical tail operation following i in the same cluster, say cluster k, will not improve the objective value if $q_i'^{(k)} > q_j'^{(k)}$.

Proof: Before the move, $C^{(k)} = t_j + p_j + q_j'^{(k)}$. After *move(i,j)*, the new completion time, $C'^{(k)} = \max(t_i, r_{i+1}) + p_i + p_\alpha + p_j + q_i'^{(k)}$ where p_α is the summation of the processing time of operations between i and j. Because $t_j = t_i + p_i + p_\alpha$; therefore, $C'^{(k)} \geq t_j + p_j + q_i'^{(k)}$. The move will not decrease the completion time of other jobs (except job k) as the lengths of their critical path do not decrease. If $q_i'^{(k)} > q_j'^{(k)}$, then $C'^{(k)} > C^{(k)}$. Therefore, the objective value will not decrease after the move.

Proposition 5: *MoveBack(j,i)* when i is an operation prior to j will not improve the objective value if j is not the first operation in the cluster.

Proof: If i is not the first operation in the cluster, the length of the critical path passing through j cannot be reduced.

Before the move, $C^{(k)} = t_i + p_i + p_\alpha + p_j + p_\beta + q_l^{(k)}$ where t_i is the starting time of operation i, $p_\alpha = \sum_{g=i+1}^{j-1} p_g$, $p_\beta = \sum_{g=j+1}^{l} p_g$, and l is the critical tail operation. After the move, $C'^{(k)} = \max(t_i, r_j) + p_j + p_i + p_\alpha + p_\beta + q_l^{(k)}$. It is obvious that the move will not improve the objective value as no job can be finished earlier.

Lemma 5: *Moveback(j,i)* where i is the operation immediately follows machine idleness and $r_j \geq t_i$ will not improve the objective value.

Proof: It follows from the proposition.

<u>CPI (Critical Path Improvement) Procedure</u>

```
(i) After applying TER/D, let Seq = current sequence, and Best = current objective value.
(ii) For $i = 1$ to $n$, where $n$ is the number of operations in the cluster. $Seq[i]$ represents the
    ith operation in the sequence. Determine the critical operations, $Cr_j$, and the number
    of critical operations, $N_{cr}$. Detect the operations immediately follows machine
    idleness, $Idle_k$, and the number of machine idleness, $N_{idle}$.
    (ii-a) For $j = 1$ to $N_{cr}$. /* moving forward */
        – NewSeq = sequence after removing operation $i$ from Seq.
        – Check precedent constraints:
            If $t_{NewSeq[j]} + p_j > t_k$, where $k$ is the precedent operation of $i$, break the loop
        – Check moving benefit:
            If $q_i^{(l)} \geq q_{Cr_j}^{(l)}$, break the loop.
            If $Obj(Move(i,Cr_j)) < Best$, insert $i$ after $j$ in NewSeq. Let Seq = NewSeq.
            Break the loop -- skip checking moving backward and do not increase $i$.
    (ii-b) For j = Nidle down to 0. /* moving backward */
        – If $t_{idle[j]-1} + p_{ideal[j]-1} < t_i$,
            NewSeq = sequence after removing operation $i$ from Seq.
            Check precedent constraints:
                If $(t_{idle[j]} < t_k)$ or $(t_{idle[j]} < r_i)$, where $k$ is the succeeding operation of $i$,
                break the loop -- increase $i$.
            Check moving benefit:
```

If $Obj(Move(i,Idle_j)) < Best$, insert i after j in $NewSeq$. Let $Seq = NewSeq$. Break the loop -- increase i.

(ii-c) If $i == Cr_{N_{cr}}$, stop.

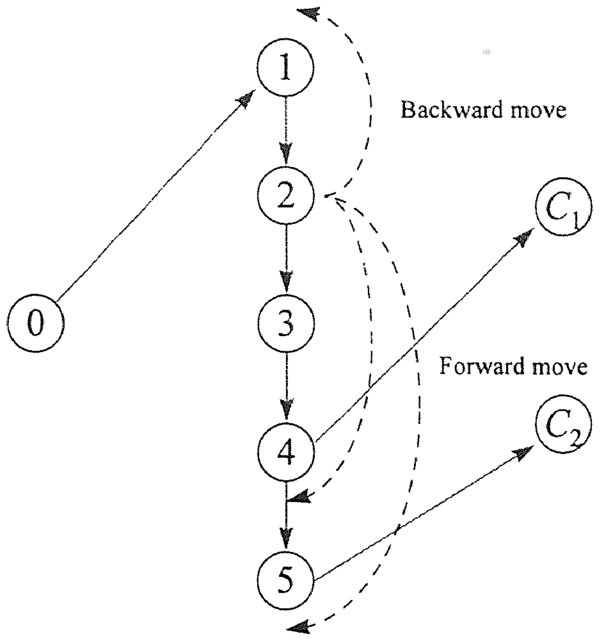


Figure 4-10: Check-points for the *Move*

CPI analyzes the critical path of the clusters. It tries to reduce the path length by moving the operations in the sequence. There are two types of moves -- forward and backward moves. The forward move is the move of the operation from the current position to follow a critical operation. Backward move is the move from the current position to the first position in the cluster. Figure 4-10 shows the critical path on a cluster. The dark gray lines represent the critical path. In order to reduce the critical path length, we check for benefit moves. In operation 2, there are three check-points – two forward moves and one backward move. The move to other positions cannot reduce the critical path length.

Example 4-1: To demonstrate the heuristic, we take a decomposed problem from Example 3-1. First, we develop the sequence using TER. Then, we apply CPI to the sequence.

Before beginning TER, the operation due dates are determined by backward assignment. For example, $d_1 = d^{(1)} - p_5 - p_3 = 19 - 6 - 6 = 7$. These values are needed for starting the heuristic and when tight indices occur. Step 1, we determine the EST of the sink nodes, $c_i^{(j)}$. These values are used to calculate the index. For example, when $c_i^{(1)}, c_i^{(2)}, c_i^{(3)} = 16, 12, \& \ 18$, the index will be

$$1 \cdot \exp\{16-19\} + 2 \cdot \exp\{12-14\} + 1 \cdot \exp\{18-25\} = 1.2716 .$$

As the indices are all equal, we select the operation with $\min(d_i - p_i)$ which is operation 2 (Table 4-1).

Table 4-1: TER -- step 1

Operation	$d_i - p_i$	$c_i^{(1)}, c_i^{(2)}, c_i^{(3)}$	*index*
1	4	16, 12, 18	1.2716
2	3	16, 12, 18	1.2716
6	3	16, 12, 18	1.2716
8	5	16, 12, 18	1.2716
11	9	16, 12, 18	1.2716
13	13	16, 12, 18	1.2716

The sequence is {2}. The new graph is generated. On step 2, operation 6 has the highest index. It is placed in the sequence which will become {2, 6}. The sequence is determined one after another until all the operations are sequenced (see

Table 4-2 - Table 4-3) The final sequence is {2, 6, 8, 1, 11, 13}. The objective value is 24.

Table 4-2: TER – step 2

Operation	Step 2		Operation	Step 3	
	$c_i^{(1)}, c_i^{(2)}, c_i^{(3)}$	*index*		$c_i^{(1)}, c_i^{(2)}, c_i^{(3)}$	*index*
1	19, 12, 18	1.2716	1	24, 10, 18	6.0375
6	16, 14, 18	2.0507			
8	16, 12, 18	0.3214	8	15, 18, 18	10.0192
11	16, 12, 11	0.3229	11	15, 10, 25	1.0549
13	16, 12, 18	0.3214	13	15, 10, 21	0.0733

Table 4-3: TER -- step 3 - end

Operation	Step 3		Operation	Step 4	
	$c_i^{(1)}, c_i^{(2)}, c_i^{(3)}$	*index*		$c_i^{(1)}, c_i^{(2)}, c_i^{(3)}$	*index*
1	24, 10, 18	6.0375	1	27,18,18	19.0009
8	15, 18, 18	10.0192			
11	15, 10, 25	1.0549	11	15, 18, 28	14.0183
13	15, 10, 21	0.0733	13	15, 18, 24	10.3862
Operation	Step 5		Operation	Step 6	
	$c_i^{(1)}, c_i^{(2)}, c_i^{(3)}$	*index*		$c_i^{(1)}, c_i^{(2)}, c_i^{(3)}$	*index*
11	27,18,31	26.00			
13	27,18,27	22.00	13	27,18,33	28.00

After the sequence generated by TER is added to the graph, the results will be the same as Figure 4-11. The critical paths are displayed in bold lines.

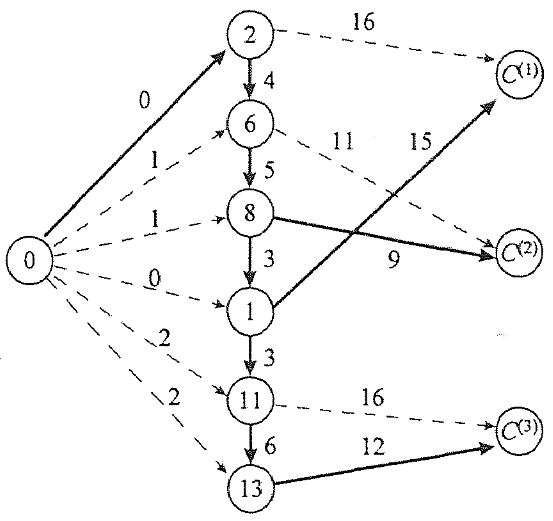


Figure 4-11: Scheduling graph after TER

CPI will check for potential improvements. First, we check the forward moves starting from the last operation in the sequence back to the first operation. As $q_{11}^{(3)} = 16 > q_{13}^{(3)} = 12$, we know that *move*(11,13) will not improve the sequence. Move(1,13) will increase the objective value to 31 similar to *move*(8,1), *move*(8,13), etc. The only move that can improve the sequence is *move*(2,8) which reduces the objective value to 20. After checking the forward moves, we check backward moves *moveback*(8,6), *moveback*(2,6), etc. There was no further improvement founded. The result {6, 8, 2, 1, 11, 13} is the result. It is also the optimal solution (Figure 4-12).

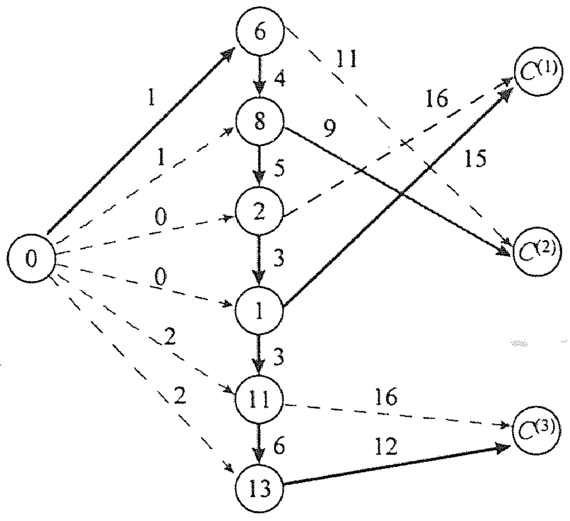


Figure 4-12: The graph after move(2,8).

4.8 Dependent Setup Time

When a machine is designed to be flexible, it may require some adjustments before processing a new task. The setup time depends on the current machine setting and the required setting for the new task. If the setup time is constant for any kind of task, we may add this setup time to the processing time. Then, the problem can be perceived as if there is no setup time.

TER with CPI technique is proved to be an efficient technique for scheduling tasks on the machine that does not require setup. If the setup time is very small comparing to the processing time, this technique is, still, proper to use. However, this technique may not provide good solutions when dependent setup time exists. To improve quality of the solution, we add local search steps to the sequence generated from TER with CPI. The local search is a random search heuristic. After it is run for a fixed number of iterations, the best solution found is reported. We strengthen the search by limiting the search space. The details are discussed below.

Local Search

The local search technique is based on interchanging two operations in the sequence. It randomly selects two operations from the basis which is generated by TER and CPI. If the newly generated sequence is feasible and provides a lower objective value, it will be used as a new basis. We limit the search for a fix number of trials. After the limit is reached, the best sequence found will be reported.

When we assume that the setup time is moderately smaller than the processing time, we may limit the interchange to be done among operations in the same cluster not including the first and the last one. This technique can improve the efficiency of the search as the search space is reduced.

4.9 Numerical Examples

We test a number of heuristics on some decomposed problems stemmed from assembly shop scheduling problems. In the following, we provide a brief description of the heuristics used.

SPT (Shortest Processing Time First): This rule selects the operation with the least processing time to be processed next. To prevent a dead-lock of the sequence, only the operations that do not have precedent constraints or all precedent constraints have been satisfied will be selected.

LPT (Longest Processing Time First): This rule will select the operation that has the highest processing time to be processed next.

FCFS (First Comes, First Serves): This rule selects the operation according its arrival time to the workstation.

EDD-O (Earliest Operation Due Date): The operation due date is determined from backward due date assignment. If the operation has two tails one to job *a* and the other to job *b*, the due date of this operation is the minimum of these two job due dates. The rule selects the operation with minimum due date to be processed next.

Modified EDD-O: It is similar to EDD-O; however, some operations are not allowed to be selected for processing. These operations are very late arrival operations. There exists at least a job that can be processed and completed before the late arrival operation enter the workstation.

EDD-J (Earliest Job Due Date): This rule selects the operation belonged to the job that has minimum due date to be processed next.

ATC (Apparent Tardiness Cost): The operation due date is determined from backward due date assignment similar to EDD-O. The index function is $I_j(t) = \frac{w_j}{p_j} \exp\left(-\frac{\max(d_j - p_j - t, 0)}{k\bar{p}} \right)$ where t is the time at which the machine became free k is the scaling parameter and $\bar{p}$ is the average processing times of the remaining jobs.

WTail (Weighted Tails): The index function is determined from the summation of the weighted tail lengths.

We have tested 50 problems generated by decomposing assembly shops with five machines. The processing time of the first 25 problems are between 3 - 10. The last 25 problems have higher processing time (5-15) and more tight due dates. It is obvious that

TER+CPI dominates other priority rules. TER, FCFS and modified EDD-O provide good sequences. Other heuristics seem to provide sequences far from optimal.

Table 4-4: Test results (SMSPT) (low utilization)

# of jobs	# of operations	Objective value (weighted tardiness)									
		SPT	LPT	FCFS	EDD-O	mod. EDD-O	EDD-J	ATC	WTail	TER	TER+ CPI
10	32	2911	2217	52	46	46	70	2682	1383	46	28
10	39	3129	4127	616	528	528	812	4812	2862	523	323
10	29	2518	2562	582	918	883	1209	2595	1180	735	576
10	33	5782	4353	1436	1225	791	1814	5822	1949	791	785
10	22	1574	2382	791	807	807	1534	1492	1081	807	791
10	32	2998	4593	0	0	0	0	3503	955	0	0
10	39	6238	7283	165	295	255	794	6899	5565	271	233
10	29	1245	3514	279	686	416	447	1346	445	357	300
10	33	5898	6947	437	531	531	923	7635	1095	434	357
10	22	4457	4233	600	434	654	968	5277	1299	434	434
10	32	2737	2260	57	50	50	142	2515	2525	50	23
10	39	5339	4891	319	408	306	788	5002	1334	329	203
10	29	2582	2804	385	475	506	954	2122	1164	529	384
10	33	5867	2948	729	957	957	1378	5950	737	682	533
10	22	3342	4118	707	792	792	950	3984	707	707	686
10	32	2039	4034	457	335	335	592	2154	764	258	85
10	39	6056	5277	1576	1679	1679	2104	4986	2349	1222	865
10	29	4158	5075	1351	3628	2200	4094	3446	3822	1734	1593
10	33	5540	6310	1939	2386	2386	3366	4929	2270	1965	1734
10	22	3277	4160	1965	2325	2325	2942	2660	1965	1965	1965
10	32	2293	2076	52	30	30	113	3205	714	0	0
10	39	6171	3698	320	789	582	1126	6268	2320	459	354
10	29	2533	2956	487	601	714	999	3544	791	749	700
10	33	6999	7286	749	871	845	1993	7868	808	749	749
10	22	1592	2608	749	853	749	988	2159	749	749	749

Table 4-5: Percent different from the minimum (SMSPT)
(low utilization)

# of jobs	# of operations	% Difference from the minimum									
		SPT	LPT	FCFS	EDD-O	mod. EDD-O	EDD-J	ATC	WTail	TER	TER+ CPI
10	32	10296	7818	86	64	64	150	9479	4839	64	0
10	39	869	1178	91	63	63	151	1390	786	62	0
10	29	337	345	1	59	53	110	351	105	28	0
10	33	637	455	83	56	1	131	642	148	1	0
10	22	99	201	0	2	2	94	89	37	2	0
10	32	-	-	-	-	-	-	-	-	-	-
10	39	3681	4314	0	79	55	381	4081	3273	64	41
10	29	346	1159	0	146	49	60	382	59	28	8
10	33	1552	1846	22	49	49	159	2039	207	22	0
10	22	927	875	38	0	51	123	1116	199	0	0
10	32	11800	9726	148	117	117	517	10835	10878	117	0
10	39	2530	2309	57	101	51	288	2364	557	62	0
10	29	572	630	0	24	32	148	453	203	38	0
10	33	1001	453	37	80	80	159	1016	38	28	0
10	22	387	500	3	15	15	38	481	3	3	0
10	32	2299	4646	438	294	294	596	2434	799	204	0
10	39	600	510	82	94	94	143	476	172	41	0
10	29	208	276	0	169	63	203	155	183	28	18
10	33	219	264	12	38	38	94	184	31	13	0
10	22	67	112	0	18	18	50	35	0	0	0
10	32	-	-	-	-	-	-	-	-	-	-
10	39	1828	1056	0	147	82	252	1859	625	43	11
10	29	420	507	0	23	47	105	628	62	54	44
10	33	834	873	0	16	13	166	950	8	0	0
10	22	113	248	0	14	0	32	188	0	0	0
Average		1810	1752	48	73	58	181	1810	1009	39	5

Table 4-6: Test results (SMSPT) (high utilization)

# jobs	# ops.	SPT	LPT	FCFS	EDD-O	mod. EDD-O	EDD-J	ATC	WTail	TER	TER+ CPI
10	32	4726	3091	284	295	295	316	4699	1103	285	158
10	39	9722	10009	1642	1599	1599	2884	8875	3457	1501	919
10	29	3493	6726	1557	2850	1810	2259	4629	2135	1768	1554
10	33	13276	11487	2156	2962	2041	5832	10954	1960	1982	1862
10	22	3131	3254	1991	2202	2202	2292	3530	2006	1991	1991
10	32	4232	5262	475	271	271	678	3869	2689	271	194
10	39	8555	9987	1446	1065	833	1298	9921	3540	831	637
10	29	5619	4155	943	1097	1103	2509	5559	1534	1013	971
10	33	10985	11488	1384	2293	1831	4449	14005	1648	1988	1482
10	22	5162	5988	1988	1988	1988	2271	5234	1988	1988	1988
10	32	3528	4852	618	582	582	738	3703	1644	465	416
10	39	5458	9384	1962	1764	1764	2106	4871	4237	1584	1279
10	29	5656	6480	1640	1839	1839	2423	5125	2031	2027	1779
10	33	6308	12165	2108	2617	2617	3560	9954	2364	2163	2084
10	22	5226	5077	2166	2166	2166	3123	6098	2339	2191	2166
10	32	3502	5637	499	556	556	846	3011	956	482	356
10	39	8011	8064	1509	2800	1656	2800	6471	2225	945	771
10	29	4422	4688	1117	2098	1696	2377	3955	1824	1260	1055
10	33	8599	6765	1270	1377	1317	3612	7896	1295	1315	1293
10	22	3266	4820	1371	1407	1407	1786	2956	1367	1407	1319
10	32	2918	4305	1103	1228	1228	1909	2082	1615	943	597
10	39	7370	8769	2112	2831	2562	4975	8224	3447	1500	1440
10	29	4983	4304	1572	3205	2250	3929	3975	4675	2627	1572
10	33	8437	6825	2810	4013	3989	5277	7448	3249	2763	2763
10	22	4665	4016	2763	2821	2763	4538	4228	2763	2763	2763

Table 4-7: Percent difference from the minimum (SMSPT)
(high utilization)

# of jobs	# of operations	% Difference from the minimum									
		SPT	LPT	FCFS	EDD-O	mod. EDD-O	EDD-J	ATC	WTail	TER	TER+ CPI
10	32	2891	1856	80	87	87	100	2874	598	80	0
10	39	958	989	79	74	74	214	866	276	63	0
10	29	125	333	0	83	16	45	198	37	14	0
10	33	613	517	16	59	10	213	488	5	6	0
10	22	57	63	0	11	11	15	77	1	0	0
10	32	2081	2612	145	40	40	249	1894	1286	40	0
10	39	1243	1468	127	67	31	104	1457	456	30	0
10	29	496	341	0	16	17	166	490	63	7	3
10	33	694	730	0	66	32	221	912	19	44	7
10	22	160	201	0	0	0	14	163	0	0	0
10	32	748	1066	49	40	40	77	790	295	12	0
10	39	327	634	53	38	38	65	281	231	24	0
10	29	245	295	0	12	12	48	213	24	24	8
10	33	203	484	1	26	26	71	378	13	4	0
10	22	141	134	0	0	0	44	182	8	1	0
10	32	884	1483	40	56	56	138	746	169	35	0
10	39	939	946	96	263	115	263	739	189	23	0
10	29	319	344	6	99	61	125	275	73	19	0
10	33	577	433	0	8	4	184	522	2	4	2
10	22	148	265	4	7	7	35	124	4	7	0
10	32	389	621	85	106	106	220	249	171	58	0
10	39	412	509	47	97	78	245	471	139	4	0
10	29	217	174	0	104	43	150	153	197	67	0
10	33	205	147	2	45	44	91	170	18	0	0
10	22	69	45	0	2	0	64	53	0	0	0
Average		606	668	33	56	38	127	591	171	23	1

The test results indicate that TER can provide a good feasible sequence. CPI can, further, improve the quality of the sequence generated by TER. It can reduce the objective value provided by the best of other priority rules on average of 14%.

Sequence Dependent Setup

We test 25 problems which are decomposed problems from the assembly shop problems. Some partial sequences on machines other than the focusing machines are added to generate delayed precedent constraints. The setup time vary from 0 - 3 while the processing time vary from 3-10 for the first 25 problems (A) and 5-15 for the last 25 problems (B). Table 4-8 to Table 4-11 show the objective values and the percent different from the minimum. For "TER+CPI+Local", 100 local searches were applied to the result from "TER+CPI".

Table 4-8: Test results (SMSPT and dependent setup) (low utilization)

# of jobs	# of operation	Objective value (weighted tardiness)								
		SPT	LPT	FCFS	EDD-O	modified EDD-O	EDD-J	TER	TER+CPI	TER+CPI +Local
10	32	3577	3911	273	484	72	712	312	53	35
10	39	8434	6965	2530	1891	1747	3452	1564	1191	1191
10	29	4883	6565	2043	2286	2238	2737	2254	2239	2159
10	34	7787	5901	2342	2624	2318	3967	2348	2290	2290
10	21	3448	4560	2668	2624	2624	3428	2624	2624	2624
10	32	4293	3454	210	326	326	473	245	76	76
10	39	6159	8975	1217	1320	1072	875	852	681	661
10	29	5305	3457	1320	1425	1320	1439	1775	1338	1320
10	34	10018	8584	1455	2332	1525	2880	1849	1452	1452
10	21	5667	5490	2332	2450	2332	3385	2332	2332	2332
10	32	3390	3615	138	59	41	173	59	18	18
10	39	6457	7719	602	512	408	708	540	435	433
10	29	3530	3093	572	1435	698	740	1125	830	830
10	34	9248	5653	1545	1526	1435	2260	1435	1435	1435
10	21	2180	4114	1534	1574	1526	1676	1526	1526	1526
10	32	4952	5808	29	63	42	171	79	18	18
10	39	8724	10478	488	677	561	1254	486	343	337
10	29	3897	2383	879	1011	730	1047	882	730	730
10	34	8929	7041	1100	1233	1023	2659	1079	1023	1023
10	21	5125	3668	1434	1233	1233	1658	1233	1233	1233
10	32	1756.93	3734.97	188	232	212	363	230	195	185
10	39	6988	8521	1145	1114	931	1263	1140	596	596
10	29	4641	4677	1132	1647	1132	1438	1186	1132	1132
10	34	8182	8768	1814	2093	1692	2638	2260	1671	1671
10	21	4129	2321	2093	2093	2093	2225	2093	2093	2093

Table 4-9: Test results (SMSPT and dependent setup) (low utilization)

# of jobs	# of operation	% Difference from the minimum								
		SPT	LPT	FCFS	EDD-O	modified EDD-O	EDD-J	TER	TER+CPI	TER+CPI +Local
10	32	10120	11074	680	1283	106	1934	791	51	0
10	39	608	485	112	59	47	190	31	0	0
10	29	139	221	0	12	10	34	10	10	6
10	34	240	158	2	15	1	73	3	0	0
10	21	31	74	2	0	0	31	0	0	0
10	32	5549	4445	176	329	329	522	222	0	0
10	39	832	1258	84	100	62	32	29	3	0
10	29	302	162	0	8	0	9	34	1	0
10	34	590	491	0	61	5	98	27	0	0
10	21	143	135	0	5	0	45	0	0	0
10	32	18733	19983	667	228	128	861	228	0	0
10	39	1483	1792	48	25	0	74	32	7	6
10	29	517	441	0	151	22	29	97	45	45
10	34	544	294	8	6	0	57	0	0	0
10	21	43	170	1	3	0	10	0	0	0
10	32	27411	32167	61	250	133	850	339	0	0
10	39	2489	3009	45	101	66	272	44	2	0
10	29	434	226	20	38	0	43	21	0	0
10	34	773	588	8	21	0	160	5	0	0
10	21	316	197	16	0	0	34	0	0	0
10	32	850	1919	2	25	15	96	24	5	0
10	39	1072	1330	92	87	56	112	91	0	0
10	29	310	313	0	45	0	27	5	0	0
10	34	390	425	9	25	1	58	35	0	0
10	21	97	11	0	0	0	6	0	0	0
Average		2961	3255	81	115	39	226	83	5	2

Table 4-10: Test results (SMSPT and dependent setup) (high utilization)

# of jobs	# of operations	Objective value (weighted tardiness)								
		SPT	LPT	FCFS	EDD-O	modified EDD-O	EDD-J	TER	TER+CPI	TER+CPI + Local
10	32	5057.85	3244	416	471	471	388	413	286	286
10	39	10109	10592	2221	2232	2232	3006	2111	1639	1629
10	29	4727	6874	2302	2427	2427	2800	2434	2449	2449
10	33	13759	12185	3045	2857	2563	6071	2609	2452	2452
10	22	3088	4333	2916	2857	2857	3230	2877	2857	2857
10	32	4471	5668.88	807	366	366	969	486	511	511
10	39	9277	10895	1871	1688	1514	1877	1473	1693	1693
10	29	7285	5307	1778	1802	1802	2612	2081	2147	2119
10	33	11920	12366	2349	2857	2762	5868	2645	2406	2406
10	22	4771	3798	2857	2857	2857	3515	2857	2857	2857
10	32	3820.9	5291	790	733	733	870	634	555	552
10	39	6377	10209	2751	2366	2366	2552	2367	1860	1747
10	29	7401	6161	2459	3373	2618	2876	2947	2938	2938
10	33	9748	16057	3399	3594	3582	4213	3456	3375	3375
10	22	6416	7046	3600	3626	3613	4454	3613	3594	3594
10	32	3840	5934	683	692	692	1036	579	317	317
10	39	8259	8731	2021	3065	1605	3122	1387	1115	1089
10	29	6115	7494	3315	3414	3228	3925	3777	3158	3158
10	33	10481	7059	3414	3414	3414	5487	4605	3414	3414
10	22	5172	6907	3414	3414	3414	3974	3535	3414	3414
10	32	3136	4638	1403	1587	1587	2193	1198	807	807
10	39	9035	8932	2776	2728	2409	4447	2439	1888	1882
10	29	5962	5478	2812	3156	2850	3833	3170	2782	2782
10	33	10741	7833	3226	3517	3172	4799	3190	3178	3156
10	22	4372	4476	3517	3526	3517	4295	3595	3595	3595

Table 4-11: Test results (SMSPT and dependent setup) (high utilization)

# of jobs	# of operations	% Difference from the minimum								
		SPT	LPT	FCFS	EDD-O	modified EDD-O	EDD-J	TER	TER+CPI	TER+CPI + Local
10	32	1668	1034	45	65	65	36	44	0	0
10	39	521	550	36	37	37	85	30	1	0
10	29	105	199	0	5	5	22	6	6	6
10	33	461	397	24	17	5	148	6	0	0
10	22	8	52	2	0	0	13	1	0	0
10	32	1122	1449	120	0	0	165	33	40	40
10	39	530	640	27	15	3	27	0	15	15
10	29	310	198	0	1	1	47	17	21	19
10	33	407	426	0	22	18	150	13	2	2
10	22	67	33	0	0	0	23	0	0	0
10	32	592	859	43	33	33	58	15	1	0
10	39	265	484	57	35	35	46	35	6	0
10	29	201	151	0	37	6	17	20	19	19
10	33	189	376	1	6	6	25	2	0	0
10	22	79	96	0	1	1	24	1	0	0
10	32	1111	1772	115	118	118	227	83	0	0
10	39	658	702	86	181	47	187	27	2	0
10	29	94	137	5	8	2	24	20	0	0
10	33	207	107	0	0	0	61	35	0	0
10	22	51	102	0	0	0	16	4	0	0
10	32	289	475	74	97	97	172	48	0	0
10	39	380	375	48	45	28	136	30	0	0
10	29	114	97	1	13	2	38	14	0	0
10	33	240	148	2	11	1	52	1	1	0
10	22	24	27	0	0	0	22	2	2	2
Average		388	435	28	30	20	73	19	5	4

CHAPTER 5

PARALLEL MACHINE SCHEDULING PROBLEM WITH TAILS (PMSPT)

Parallel machine workstations are widely found in practices. A single machine may not be able to complete the large amount of tasks within the time limit. Therefore, similar machines are added to increase the throughput. In most cases, double the number of machines does not double the production. The synchronizing of the machines effects the output level. Tremendous amounts of research efforts have been spent on finding scheduling techniques for this type of workstations. The majority of the research is done on optimizing the machine sequences on makespan and flow time (see Pinedo, 1995).

The problem that we are looking at stems from the decomposed sub-problems of the network problem. First, the model for the simple case where there is no setup time and all machines are identical is presented. We propose two heuristics on solving this problem. The branch and bound method is developed to find the optimal solution for the comparison purpose. Then, we extend both heuristics to cover the dependent setup time and the workstation with different speed machines.

5.1 Mathematical Formulation

To develop a mathematical formulation, two fictitious jobs 0 and n+1 with $p_0 = p_{n+1} = 0$ are added. Let $x_{i,j}^{k} = 1$ if operation i is scheduled immediately before operation j on machine k, and 0 otherwise. Similarly, let $y_{i,j}^{k} = 1$ if operation i is scheduled before j on machine k. The formulation is as follows.

$$\text{Min} \sum_{j=1}^{n} \left\{ w^{(j)} T^{(j)} - h^{(j)} E^{(j)} \right\},$$

subject to

(1) $t_i \geq r_i,$ $i = 1, ..., m,$

(2) $t_{i'} \geq \sum_{k=1}^{c} x_{i,i'}^{k} (t_i + p_i),$ $i' = 1, ..., m, i = 1...m,$

(3) $t_{i'} \geq t_i + p_{ii'},$ $(i, i') \in Q,$

(4) $C^{(j)} \geq t_i + q_i^{(j)},$ $j = 1, ..., n, i = 1...m,$

(5) $\sum_{i=1}^{n} \sum_{k=1}^{m} x_{i,i'}^{k} = 1,$ $k = 1, ..., c,$

(6) $\sum_{i'=1}^{n} x_{0,i'}^{k} = 1,$ $k = 1, ..., c,$

(7) $\sum_{i=1}^{n} x_{i,n+1}^{k} = 1,$ $k = 1, ..., c,$

(8) $y_{i,i'}^{k} \geq x_{i,i'}^{k},$ $i = 0, ..., m+1, i' = 0, ..., m+1,$

$k = 1, ..., c,$

(9) $y_{i'',i}^{k} \cdot y_{i,i'}^{k} \leq y_{i'',i'}^{k},$ $i'' = 0, ..., m, i = 1, ..., m,$

$i' = 1, ..., m+1, k = 1, ..., c,$

(10) $y_{i,i'}^{k} + y_{i',i}^{k} = 1,$ $i = 0, ..., m+1, i' = 0, ..., m+1,$

(11) $x_{i,i}^{k} = 0,$ $i = 0, ..., m+1, k = 1, ..., c,$

(12) $t_0 = 0,$

(13) $x_{i,i'}^{k} \in \{0,1\},\ y_{i,i'}^{k} \geq 0,$

(14) $L^{(j)} = C^{(j)} - d^{(j)},$ $\quad j = 1, \ldots, n,$

(15) $T^{(j)} \geq 0;\ T^{(j)} \geq L^{(j)},$ $\quad j = 1, \ldots, n,$

(16) $E^{(j)} \geq 0;\ E^{(j)} \geq -L^{(j)},$ $\quad j = 1, \ldots, n,$

where,

t_i	Starting time of operation *i,*
r_i	release time of operation *i,*
p_i	processing time of operation *i,*
$p_{ii'}$	time precedent relation between operation *i* and *i′* (time that operation *i′* need to wait after operation *i* has completed),
$C^{(j)}$	completion time of job *j,*
$q_i^{(j)}$	remaining processing time for job *j* after operation *i* has completed,
$L^{(j)}$	lateness of job *j,*
$d^{(j)}$	due date of job *j,*
$T^{(j)}$	tardiness of job *j,*
$E^{(j)}$	earliness of job *j.*

Constraint Interpretation

(1) Operation release time constraint

(2) Machine can process one job at a time.

(3) Operation precedent constraint

(4) Determine completion time

(5) Except operation 0, every operation has, exactly, one prior operation in

the processing sequence.

(6) For operation 0, there is, exactly, one link to each machine. ($n>m$)

(7) There is, exactly, one link from each machine to operation $n+1$.

(8) If j is process immediately after i, then operation j is scheduled after operation i.

(9) If operation h is processed before i and i before j, then h is processed before j.

(10) If j is scheduled after i, it cannot be processed before i.

(11) - (13) Basic constraints.

(14) - (16) Determine the variables for objective function evaluation.

5.2 Problem Analysis

To schedule the parallel machines workstation, the task can be separated into two steps -- workload assignment and operation sequencing. The first step is to allocate tasks (operations) to machines. Poor workload assignment leads to unequal machine utilization. Operation sequencing step is to determine the processing order of the operations assigned to the machine. In order to satisfy the objective, both steps should be done in parallel.

5.3 Sequence Graph

The disjunctive graph that has been used for single machine scheduling problem can be modified to include the parallel machines problem. Nodes, conjunctive arcs, and disjunctive arcs are created in the same manner. However, the objective of the problem is a little different. Instead of finding a non-cycle selection from the clique, it is to partition the clique into n cliques and find a non-cycle selection for each clique, where n is to the number of machines in the workstation. Figure 5-1 shows a non-cyclic selection on a clique. This selection represents a feasible sequence for a single machine workstation. Figure 5-2 shows the same sub-graph partitioned into two cliques. Two non-cyclic selections represent a feasible sequence for a 2-parallel machines workstation.

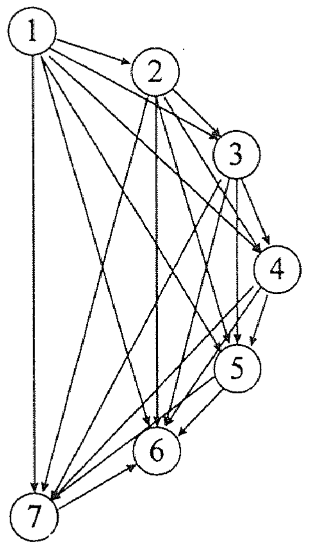


Figure 5-1: A non-cyclic selection for a single machine workstation

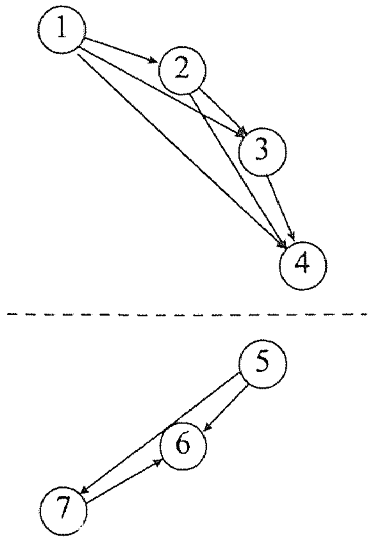


Figure 5-2: Two non-cyclic selections for a 2-parallel machines workstation

For the *m* parallel machines workstation, *m* selections should be determined from the clique associated to the workstation. Each of them should not create a cycle. After the selections are entered in the graph, the job completion time can be determined.

<u>Parallel Machines with Different Speeds</u> : When the speed of the machines are not equal but in ratio, further modification is necessary. The outbound arc length from node *i* to *i'* in the same clique will be changed to $\frac{p_i}{f_k} + s_{ii'}$, where f_k is the speed of machine *k* assigned to process operation *i*, and $\frac{p_i}{f_k}$ if *i* and *i'* are not in the same clique.

Parallel Unrelated Machines : For the most general case where the processing time of the tasks depend on the processing machine and is not constant, the outbound arc lengths from node i to node i' in the same clique are modified to $p_i(k) + s_{ii'}$ where $p_i(k)$ is the processing time of operation i on machine k. If i and i' are not in the same clique, the setup time will not be included and the length is $p_i(k)$.

5.4 PEDD-O (Parallel Earliest Operation Due Date)

This priority rule is the extension of EDD rule. PEDD-O does not only assign operations to the machines but also sequence the operations on each machine. It assigns the operations according to their indices. These indices are determined from the length of the tails, the job due dates, and the delayed precedent lengths. After the next operation to be sequenced has been determined, it is assigned to the first available machine.

(i) Let U be a set of unscheduled jobs; $A = \varnothing$; $t = 0$; $G =$ directed graph; $l = 0$;

$rl_k = r_k$ for $k = 1, \ldots, m$; machine available time, $a_k = 0$ for $k = 1, \ldots, c$.

(ii) $A = A \cup \{j, rl_j \leq t; j \in U\}$; $U = U - \{j\}$. If $A = \varnothing$ and $U = \varnothing$, stop.

Else, if $A = \varnothing$ and $U \neq \varnothing$, $A = A \cup \{j, rl_j = \min_{k \in U} rl_k; j \in U\}$; $U = U - \{j\}$.

(iii) Schedule $\{j\}$ on machine mc such that $j \in A$ and $I_j \leq I_k$; $k \in A$.

$A = A - \{j\}$. $t_j = \max \{rl_j, t\}$. $t = \max \{rl_j, t\} + p_j$. $G = G \cup (l,j)$ on machine mc.

$rl_k = t + p_{jk}$ if $(j,k) \in Q$. $l = j$; $a_{mc} = \max(r_j, a_{mc}) + p_j$.

(iv) Goto (ii).

5.4.1 Determine Index and Machine Assignment

To determine the index for operation *j*, we, first, need to assign the operations to the machines. Machines in the workstation may have different available time. Some of them need to complete the operations assigned in the previous iterations before they become available again. We select machine *mc* that $\min\limits_{k=1,\dots,c}\left(\max(r_i, a_k) + p_i\right)$.

Given graph *G* which is the directed graph *D* with partial selection, the index for operation *i'* is determined by adding the conjunctive arc (*i*, *i'*) where *i* is the previously sequenced operation on machine *mc*. Then, the EST of sinks are recalculated. The index value is determined from the index function $I_i = \min(d^{(j)} - q_i^{(j)}, I_{i''} - p_{i'i''})$ for $j = 1, \dots, n$; $i'' = 1, \dots, m$.

5.4.2 Improving PEDD-O

After applying PEDD-O, we obtain a list of sequences for all machines in the workstation. These sequences can be adopted as the operation assignment. Subsequently, the problem is reduced to *m* single machine scheduling problems. These problems can be solved by the heuristic developed in the previous chapter. We can improve the sequences generated by PEDD-O by applying CPI method on the sequence in each machine.

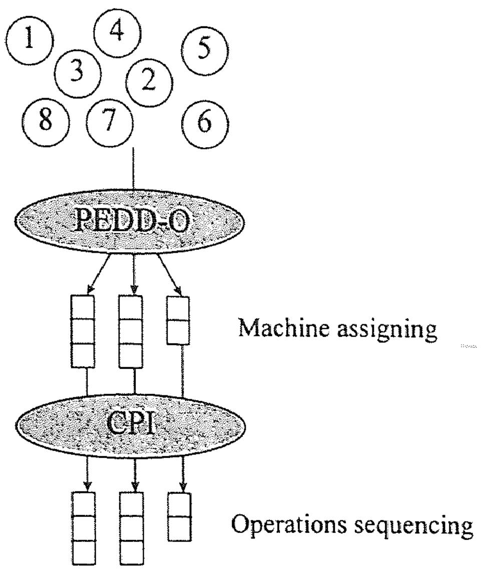


Figure 5-3: Steps in the heuristic

5.5 Beam Search

Beam search is based on the idea of branch and bound. The major distinction from branch and bound is that not all nodes at any given level are evaluated. Only the most promising *k* nodes will be selected as the nodes to branch from. The rests will be permanently discarded. The number of promising nodes to branch at each level is fixed to limit the search space. It is defined as *beam width.* The potential nodes are selected from an evaluation process.

Definition: *Parallel sequence* is a set of machine sequences for all machines in the workstation.

Definition: *Linear sequence* is a sequence representation where all elements in the sequences are lined up into a string. If operation *i* precedes operation *j* in the linear sequence, *i* is processed before or at the same time as *j*.

5.5.1 Linear Sequence ↔ Parallel Sequence Transformation

Providing a machine assignment scheme, the linear sequence can be transformed to parallel machine sequences and backwards. For example, a list of sequences for parallel machines can be generated by assigning the operation in the linear sequence to the first available machine. After assigning the operations to the machine, update the machine available time. This assignment scheme will transform the linear sequence to a parallel sequence. The reverse can be done by assigning the operations on parallel machines back to a linear sequence according to their starting time. The one-to-one relationship can be developed.

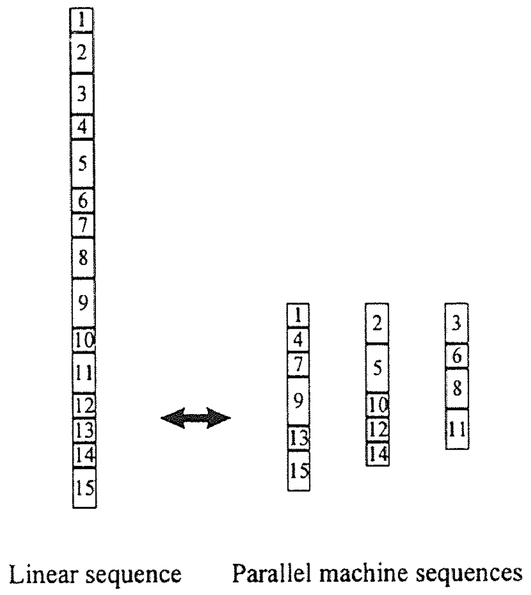


Figure 5-4: Linear seqeunce vs. parallel machine sequences

Proposition 1: For any parallel sequence, there exists at least a linear sequence that is equivalent or dominates it.

Proof: Suppose there exists a parallel sequence $S = \{\{o_{11}, o_{12}, o_{13},\ldots, o_{1n_1}\}$, {o21, o22, o23,..., $o_{2n_2}\},\ldots, \{om_1, om_2, om_3,\ldots, o_{mn_m}\}\}$ that cannot be represented by a linear sequence. Let $S_j = \{oj1, oj2,\ldots, ojk, ojk+1,\ldots,o_{jn_j}\}$ be the sequence for machine j in S

and $e = \{ o_{jk},\ldots,o_{jn_j} \}$ be the section that cannot be converted to a linear sequence. By removing e from S, a linear sequence can be generated. Let ${}^{l}S' = L(S - e) + e$, where $L(s)$ is the linear sequence transformation from parallel sequence s. The starting time of operations in $S - e$ remains the same as in S. The completion time of $o_{jk},\ldots,o_{jn_j}$ in S', where ${}^{l}S' = L(S')$, should be decreased or at least equivalent to S. Therefore, ${}^{l}S'$ is equivalent or dominates S. Figure 5-5 and Figure 5-6 show the conversion from S to S'.

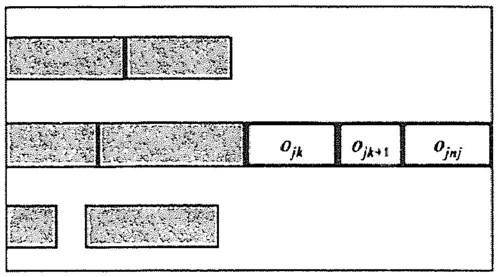


Figure 5-5: Parallel sequence S that cannot be represented by linear sequence

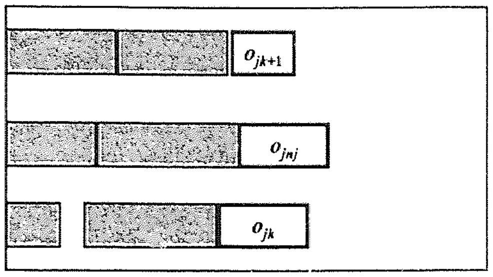


Figure 5-6: Parallel sequence S' after the conversion

As we can acquire parallel machine sequences from a linear sequence, the search will be done on the linear sequence. This technique reduces the search space drastically. In the heuristic, we use the machine assignment scheme that allocates the operation in the linear sequence to the machine that can complete the processing earliest.

5.5.2 Branching

The heuristic starts from an empty sequence node. An active operation set is determined to avoid the infeasible branching. The set members are active operations which are the operations that do not have precedent constraints or the constraints are satisfied. The late arrival operations are removed from the set. The late arrival operation is the operation that its release time is greater than the completion time of some operations, $r_i >$ $\min(\max(t,r_{i'})+p_{i'})$ for all $i' \in$ unscheduled operations where t is the earliest completion time of last sequenced operations on machines ("t" can be perceived as the next machine available time). First, the starting node will be branched according to the elements in the set. The linear sequences are generated. These sequences will be transformed to parallel machine sequences according to the machine assignment scheme.

5.5.3 Evaluating

After the branching, partial linear sequences are obtained. PEDD-O is applied to complete these sequences. The objective values of these completed sequences provide a measurement of the potential of these partial sequences. The best k nodes are kept. The rests will be discarded. These k nodes will be branched on the next step. Each step, only k nodes are kept to limit the searching space. The procedure continues until the completed sequence is obtained.

Example 5-1: We use Example 3-1 to demonstrate the beam search with beam width equals three. There are two identical machines in the cutting workstation,. The information on the operations is provided below.

Starting from S=∅, the first branching generates six nodes (Figure 5-7). The potential values are determined by completing the partial sequence with PEDD-O. For example, the second node with sequence {M1:2; M2:∅} has potential value = 1.98 which is from the sequence {M1:2,1,11; M2:6,8,13}. The first three nodes are selected for the branching in level 2. The branching continues until the set is empty (Figure 5-8 and Figure 5-9). The final sequence is {M1:1,6,13; M2:2,8,11} which provides objective value of -0.05.

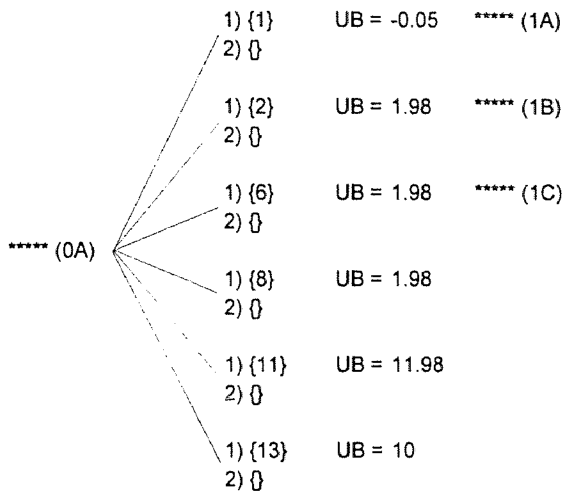


Figure 5-7: Beam search level 1

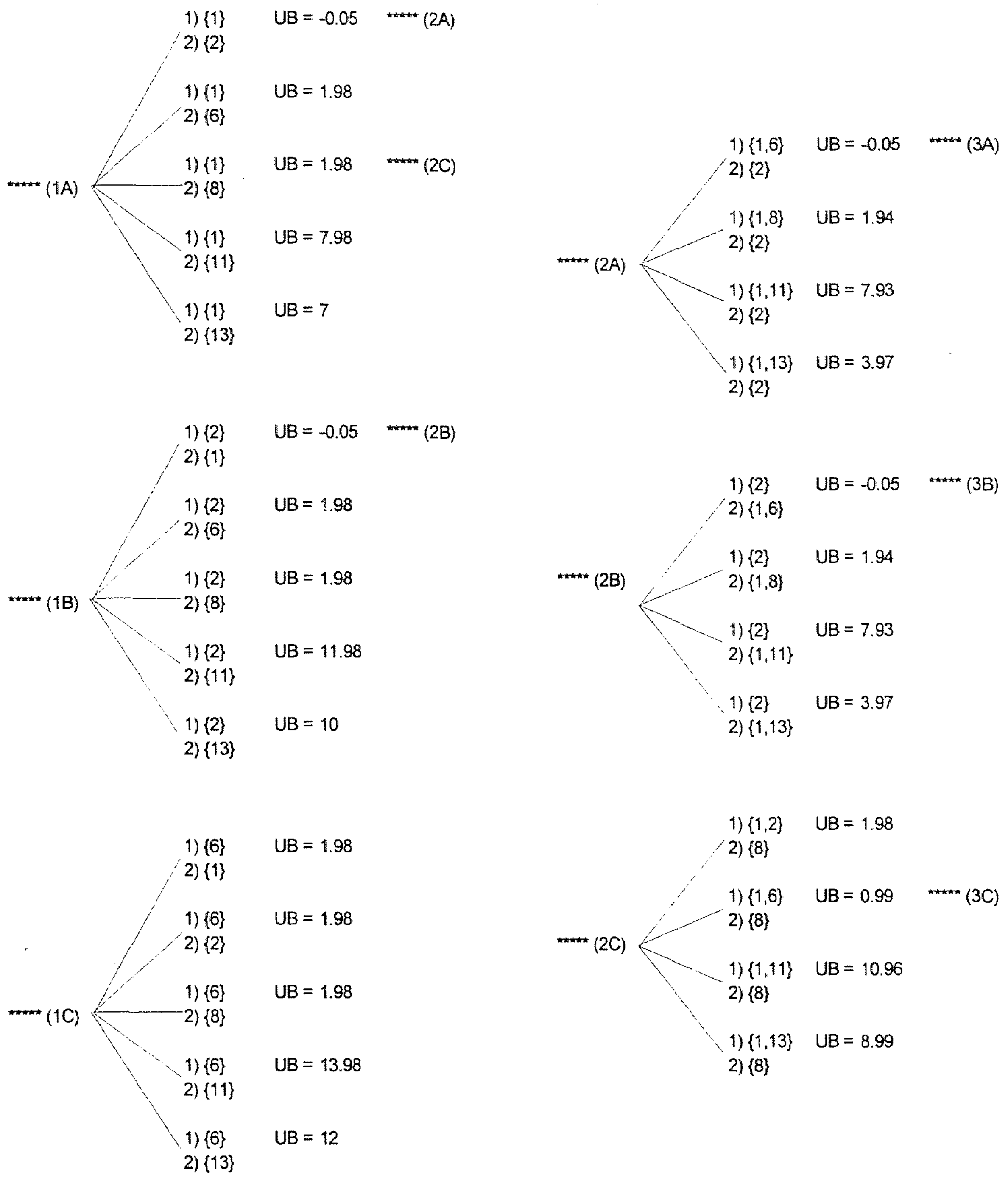

***** (1A)
1) {1} 2) {2} UB = -0.05 ***** (2A)
1) {1} 2) {6} UB = 1.98
1) {1} 2) {8} UB = 1.98 ***** (2C)
1) {1} 2) {11} UB = 7.98
1) {1} 2) {13} UB = 7
***** (1B)
1) {2} 2) {1} UB = -0.05 ***** (2B)
1) {2} 2) {6} UB = 1.98
1) {2} 2) {8} UB = 1.98
1) {2} 2) {11} UB = 11.98
1) {2} 2) {13} UB = 10
***** (1C)
1) {6} 2) {1} UB = 1.98
1) {6} 2) {2} UB = 1.98
1) {6} 2) {8} UB = 1.98
1) {6} 2) {11} UB = 13.98
1) {6} 2) {13} UB = 12
***** (2A)
1) {1,6} 2) {2} UB = -0.05 ***** (3A)
1) {1,8} 2) {2} UB = 1.94
1) {1,11} 2) {2} UB = 7.93
1) {1,13} 2) {2} UB = 3.97
***** (2B)
1) {2} 2) {1,6} UB = -0.05 ***** (3B)
1) {2} 2) {1,8} UB = 1.94
1) {2} 2) {1,11} UB = 7.93
1) {2} 2) {1,13} UB = 3.97
***** (2C)
1) {1,2} 2) {8} UB = 1.98
1) {1,6} 2) {8} UB = 0.99 ***** (3C)
1) {1,11} 2) {8} UB = 10.96
1) {1,13} 2) {8} UB = 8.99

Figure 5-8: Beam search level 2-3

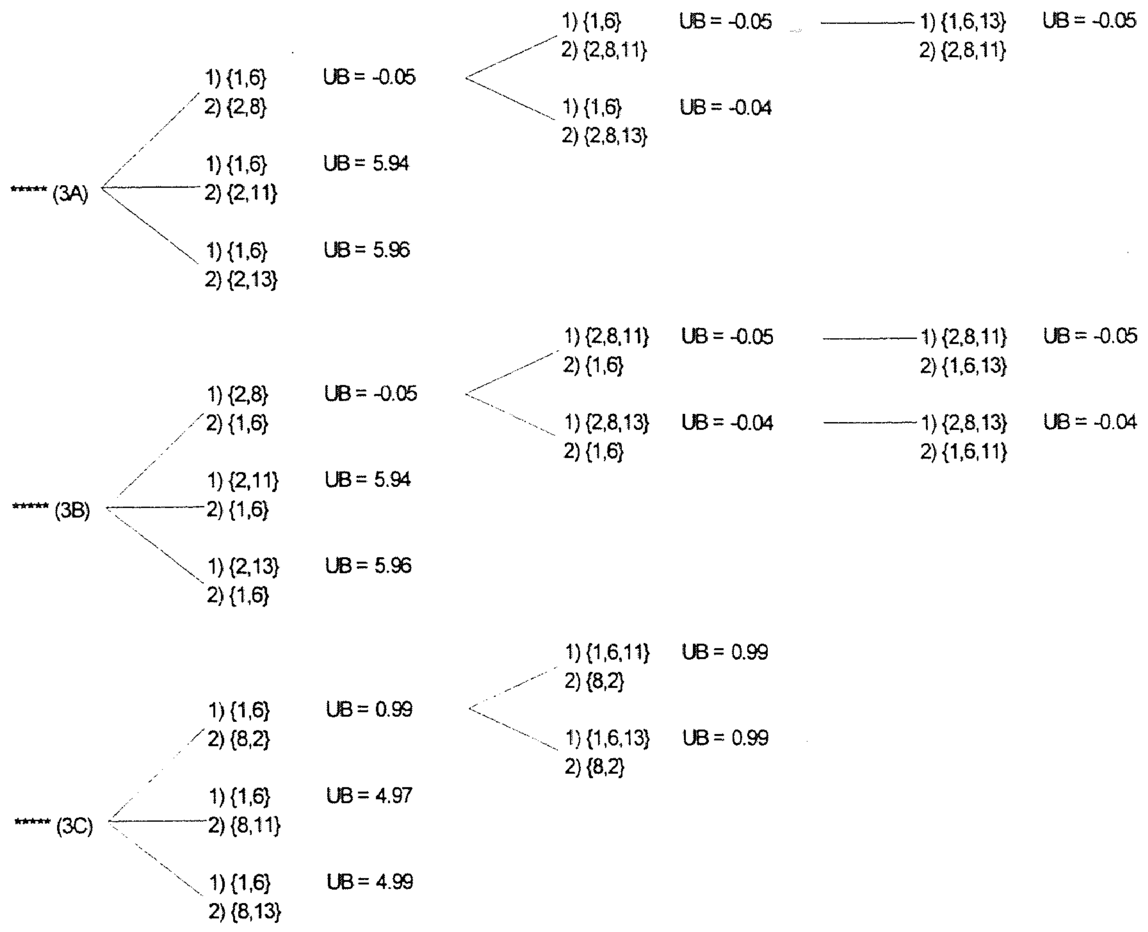

1) {1,6} 2) {2,8} UB = -0.05
1) {1,6} 2) {2,11} UB = 5.94
1) {1,6} 2) {2,13} UB = 5.96
***** (3A)
1) {1,6} 2) {2,8,11} UB = -0.05
1) {1,6} 2) {2,8,13} UB = -0.04
1) {1,6,13} 2) {2,8,11} UB = -0.05
***** (3B)
1) {2,8} 2) {1,6} UB = -0.05
1) {2,11} 2) {1,6} UB = 5.94
1) {2,13} 2) {1,6} UB = 5.96
1) {2,8,11} 2) {1,6} UB = -0.05
1) {2,8,13} 2) {1,6} UB = -0.04
1) {2,8,11} 2) {1,6,13} UB = -0.05
1) {2,8,13} 2) {1,6,11} UB = -0.04
***** (3C)
1) {1,6} 2) {8,2} UB = 0.99
1) {1,6} 2) {8,11} UB = 4.97
1) {1,6} 2) {8,13} UB = 4.99
1) {1,6,11} 2) {8,2} UB = 0.99
1) {1,6,13} 2) {8,2} UB = 0.99

Figure 5-9: Beam search level 4-6

The computation complexity of this heuristic is $O(kmn^3)$ where k is the beam width, m is the number of jobs and n is the number of operations.

5.6 Sequence Dependent Setup and Related Machines

The related machines are machines in the same workstation that have different processing speeds. However, the ratios of the speeds are fixed for any task. For example, machine A can complete the processing of task j in 5 minutes while it requires 10 minutes on machine B. If machine A and B are related machines and the processing time for task k on machine A is 7 minutes, the processing time of the same task on machine B should be 14 minutes.

5.6.1 Extended PEDD-O

When parallel machines are concerned, PEDD-O need a slight modification on the machine assignment step. The details are discussed below. After the operations are assigned to the machines, we apply CPI and followed by local searches to improve the sequence on each machine.

Determine Index and Machine Assignment

To determine the index for operation j, we, first, need to assign the operation to the machine. Machines in the workstation may have different available time. Some of them need to complete the operations assigned in the previous iterations before they become available again. The problem is more complex when processing speed of each machine is not equal and the dependent setup time is concerned. We select machine *mc* that can complete the task earliest among the machines in the workstation.

Given graph G which is the directed graph D with partial selection, the index for operation i' is determined by adding the conjunctive arc (i,i') where i is the previously sequenced operation on machine *mc*. If the processing speed on the machine differs from the average, we need to update the weight of disjunctive arcs originated from i'. Then, the EST of sinks are recalculated. The index value is determined from the index function $I_i = \sum_j w^{(j)}(T^{(j)} + e^{E^{(j)}})$, where $T^{(j)} = \max(0, C^{(j)} - d^{(j)})$, $E^{(j)} = \max(0, d^{(j)} - C^{(j)})$, and $C^{(j)}$ is the EST of sink j.

5.6.2 Beam Search

The modification of the beam search is very small. We improve the evaluation process by using PEDD-O/w Setup in place of PEDD-O. The branching scheme remains the same.

5.7 Numerical Example

We test the heuristics with 30 problems generated from decomposing an assembly shop with five jobs on five machines each job has ten operations. Each problem has 16 operations. The results indicate that PEDD-O + CPI can provide sequences better than all priority rules that we have tested (see Table 5-1). However, Beam search with beam width 3 can outperform all other methods.

Table 5-1: Test results (PMSPT)

Problem	Beam(3)	PEDD-O + CPI	FCFS	EDD-J	SPT	LPT	ATC(2,2)	ATC(1,1)
1	-0.18	-0.08	64.67	167.91	440.92	608.94	497.00	554.00
2	21.97	28.00	118.00	379.97	958.00	906.00	949.00	1000.00
3	225.99	268.00	301.00	545.98	800.00	975.00	1106.00	1040.00
4	9.93	22.97	93.95	348.00	406.94	783.00	707.00	707.00
5	216.00	255.00	515.00	658.00	723.00	1464.00	1412.00	1412.00
6	492.00	495.00	687.00	869.00	895.00	1074.00	1412.00	1399.00
7	-0.30	-0.29	34.85	112.93	393.00	313.99	376.00	376.00
8	193.93	193.93	310.99	328.93	585.00	912.00	978.00	978.00
9	208.93	208.93	335.00	436.93	765.00	894.00	836.00	836.00
10	-0.34	-0.17	-0.31	260.00	681.00	1329.00	1085.00	1085.00
11	7.95	7.95	60.99	450.00	1227.00	1654.00	1626.00	1626.00
12	115.99	129.99	207.00	431.00	1183.00	1810.00	1525.00	1411.00
13	-0.49	-0.42	-0.25	169.97	345.93	226.00	268.00	268.00
14	72.93	72.93	166.00	406.00	966.00	666.00	1046.00	1046.00
15	284.99	287.00	422.00	834.00	1226.00	1246.00	1246.00	1246.00
16	-0.34	-0.32	-0.35	-0.07	495.97	425.00	623.99	558.97
17	-0.25	-0.25	-0.20	16.99	1109.00	863.00	1199.00	1109.00
18	101.93	101.95	110.95	245.00	1165.00	660.00	1093.00	1147.00
19	-0.27	-0.23	-0.25	5.89	667.99	187.98	504.00	504.00
20	-0.15	-0.15	-0.15	84.00	1022.00	784.00	1106.00	1106.00
21	43.99	43.99	43.99	230.00	1098.00	958.00	1378.00	1378.00
22	-0.44	-0.39	-0.38	61.87	370.00	207.96	324.00	357.00
23	-0.23	-0.23	37.83	189.87	712.00	834.00	1002.00	1002.00
24	-0.21	-0.21	50.84	90.87	666.00	873.00	999.00	1027.00
25	-0.18	-0.11	3.83	238.92	359.00	244.96	521.00	509.00
26	147.90	180.91	240.91	386.94	752.00	1173.00	1285.00	1253.00
27	289.00	407.00	401.00	436.00	789.00	1532.00	1358.00	1358.00
28	-0.43	-0.38	-0.33	88.95	572.99	610.00	656.00	666.00
29	4.84	14.90	36.93	242.98	1305.00	1161.00	1193.00	1193.00
30	173.98	174.00	173.98	349.98	1277.00	966.00	1062.00	999.00
average	86.95	96.31	147.15	302.23	798.56	878.06	979.10	971.70

<u>Sequence Dependent Setup</u>

The similar tests are performed on the problems with sequence dependent setup. Adding Local search in the evaluation steps of the Beam search does not significantly improve the performance of the heuristic. Only the results from problem 13 show a significant improvement.

Table5-2: Test results (parallel machines with dependent setup)

Problem	Beam(3) Local(50)	Beam(3)	PEDD-O + CPI	EDD-O	FCFS	PTER	EDD-J	SPT	LPT	ATC(2,2)	ATC(1,1)
1	-0.35	-0.34	-0.32	-0.32	-0.35	-0.30	3.97	507.98	461.00	606.00	653.00
2	-0.23	-0.23	-0.23	-0.23	-0.19	-0.19	48.99	1150.00	890.00	1402.00	1312.00
3	203.99	203.99	203.99	203.99	242.99	496.00	505.00	1445.00	924.00	1377.00	1411.00
4	-0.27	-0.27	-0.23	-0.23	-0.24	86.98	11.89	709.00	192.99	488.00	488.00
5	5.88	5.88	5.88	5.88	5.88	189.00	149.00	1087.00	821.00	1115.00	1115.00
6	50.98	50.98	50.98	50.98	99.00	183.00	267.00	1023.00	1065.00	1331.00	1331.00
7	-0.40	-0.43	-0.34	-0.34	-0.38	-0.32	76.87	429.00	271.97	486.00	402.00
8	4.80	4.80	4.80	4.80	45.84	23.83	238.87	772.00	912.00	1108.00	1094.00
9	84.00	84.00	84.00	84.00	95.00	84.00	233.00	543.00	882.00	1190.00	1218.00
10	-0.18	-0.17	7.90	7.90	19.85	23.89	259.92	398.00	329.97	527.00	503.00
11	213.91	213.91	239.91	239.91	374.91	374.91	589.94	811.00	1404.00	1308.00	1260.00
12	-0.38	-0.38	-0.32	-0.32	-0.31	89.73	121.96	572.99	623.00	668.00	704.00
13	24.86	29.88	34.93	34.93	72.96	464.00	286.98	1376.00	1232.00	1237.00	1237.00
14	222.97	222.98	232.00	232.00	211.97	473.00	376.97	1480.00	1087.00	1416.00	1416.00
average	57.83	58.18	61.64	61.64	83.35	177.68	226.45	878.85	792.57	1018.50	1010.29

CHAPTER 6

BATCH MACHINE SCHEDULING PROBLEM WITH TAILS (BMSPT)

Besides single machine and parallel machine workstations, there is a special type of machine that can process more than one job in a single run. This type of machine is called "Batch Machine". The batch machine can be founded in various production facilities such as testing equipment for electronic circuits, paint booth, heat treatment equipment, etc. It violates the basic assumption on the machine in the classical job shop scheduling problem that two or more jobs cannot be processed in the same instance on a machine.

There are a few researches on batch machine scheduling in job-shop environment. The majority of the batch machine researches were conducted in chemical engineering where job is not discrete and the product transfer rate is significant. The work by Lee *et al.* (1992) has shown the significance of this type of model. They developed heuristics to tackle the batch machine scheduling problem on minimizing the maximum lateness in an electronic circuit testing facility.

Our objective is to find the sequence that minimizes the weighted tardiness & earliness. Two models are presented. The first one considers the case that $p_i = p$ and $s_{ij} = 0$. Next, we extend the heuristic for the more general case where the processing time of each task is not equal and dependent setup time is considered.

6.1 Mathematical Formulation

The following formulation is constructed for the batch machine scheduling problem with tails when $p_i = p$ and the machine setup time is negligible. The details of this problem can be found in Chapter 3.

$$\text{Min} \sum_{j=1}^{n} \left(w^{(j)} T^{(j)} - h^{(j)} E^{(j)} \right),$$

subject to

(1) $t^b \geq x_i^b r_i$, for $i = 1, \ldots, m$; $b = 1, \ldots, m$,

(2) $C^{(j)} \geq x_i^b \left(t^b + p + q_i^{(j)} \right)$, for $j = 1, \ldots, n$; $i = 1, \ldots, m$,

(3) $C^{(j)} \geq rc^{(j)}$, for $j = 1, \ldots, n$.

(4) $t^b \geq t^{b'} + p$ or $t^{b'} \geq t^b + p$, for all b and b',

(5) $t^b \geq x_i^b x_{i'}^{b'} \left(t^{b'} + p_{ii'} \right)$, for $(i,i') \in Q$,

(6) $L^{(j)} = C^{(j)} - d^{(j)}$, for $j = 1, \ldots, n$,

(7) $T^{(j)} \geq 0$ and $T^{(j)} \geq L^{(j)}$, for $j = 1, \ldots, n$,

(8) $E^{(j)} \geq 0$ and $E^{(j)} \geq -L^{(j)}$, for $j = 1, \ldots, n$,

(9) $\sum_{b=1}^{m} x_i^b = 1$, for $i = 1, \ldots, m$,

(10) $\sum_{i=1}^{m} x_i^b \leq z$, for $b = 1, \ldots, m$,

(11) $x_i^b = 0$ or 1,

where,

$L^{(j)}$ is the lateness of job j,

$T^{(j)}$ is the tardiness of job j,

$E^{(j)}$ is the earliness of job j,

x_i^b is 1 if task i is assigned to batch b or 0 otherwise,

t^b is the starting time of batch b.

This mathematical formulation is somewhat similar to the single machine workstation problem. The major different is the batching constraints, (9) - (11). Constraint (9) states that the task must be assigned to a batch while constraint (10) limits the batch size to z. We use integer 0/1 variable, x_i^b, to identify the task-batch assignment.

6.2 Scheduling Graph Representation

The above problem can be represented as a graph problem. In the graph, there are m starting nodes representing m tasks. The source node, 0, is connected to all m nodes with arc weights equal to the task release time. There are m completion nodes matching with the m starting nodes. Each completion node, i, represents the completion time of the task. It is connected to sink nodes, j, which representing the job completion with weight $q_i^{(j)}$. The number of sinks is equal to the number of jobs which is n. The start node and completion node are connected with an arc weighted p. There are arcs with weight $rc^{(j)}$ connecting the source with the sink j. Figure 6-1 shows the unscheduled graph with four jobs and eight operations.

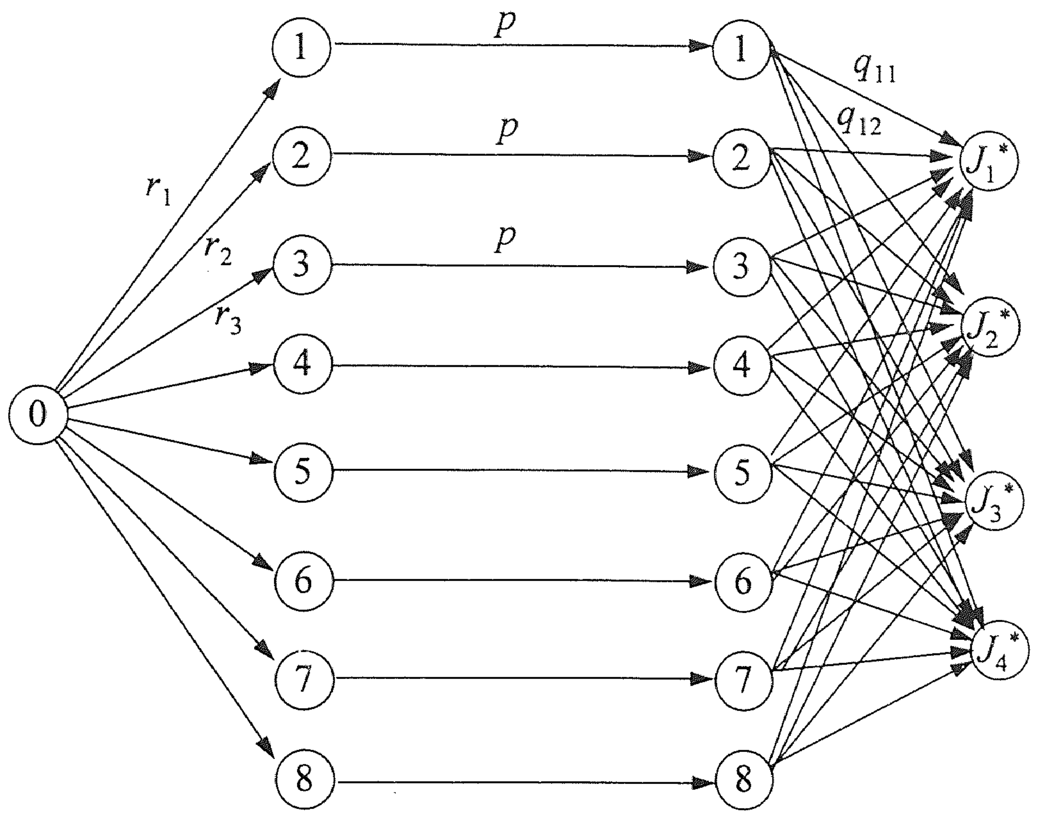


Figure 6-1: Unscheduled graph

The objective of the problem is to insert dummy nodes in between the starting nodes and completion nodes that will minimize the objective function. These dummy nodes represent the batches. The inserted dummy node has at most z arcs connecting to the completion nodes and the same number of arcs connecting from the starting nodes (see Figure 6-2). The arcs going into the dummy node have zero weight. The arcs going out off it have weight p, the batch processing time. All the starting and completion nodes must be connected to dummy nodes in order to obtain a complete sequenced graph. Figure 6-3 shows the complete sequenced graph. The processing is separated into three batches. The first two batches process three operations in a single run while the last batch process the remaining two operations. The objective function of the graph is to minimize $\sum_{i=1}^{m} w_i T_i - h_i E_i$ which is similar to the original problem. The starting time of sink node j is comparable to the completion time of job j. It can be easily determined using CPM in project scheduling technique. After the job completion time is found, the objective function can be determined.

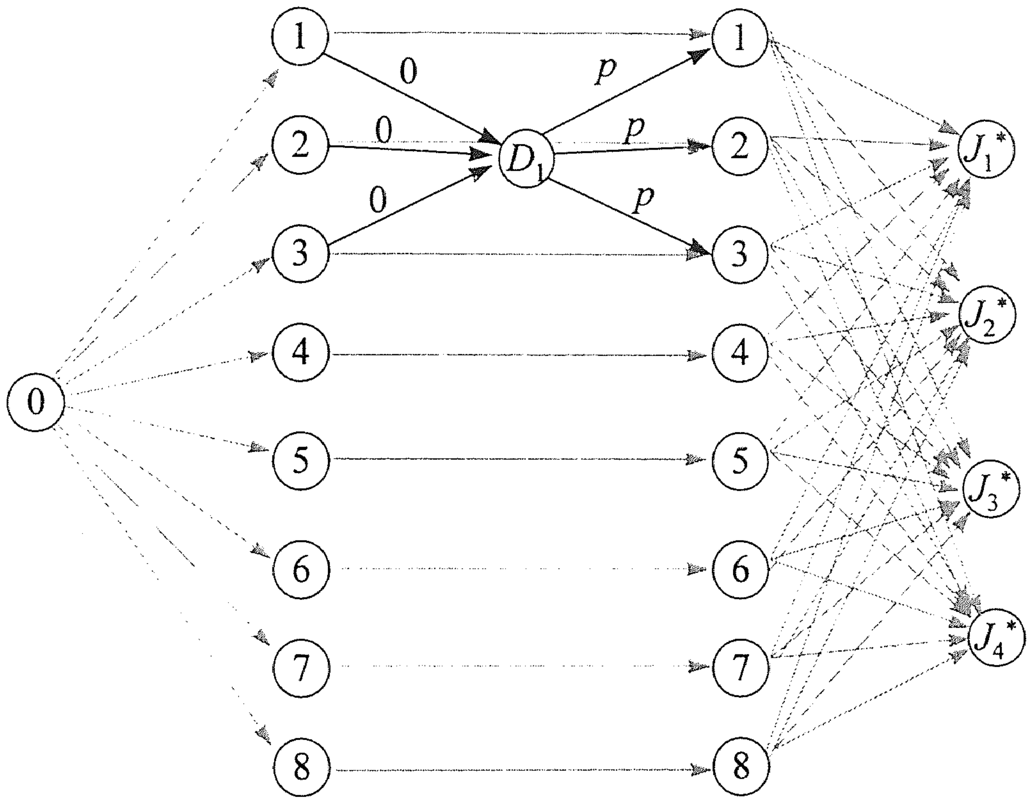


Figure 6-2: Adding a dummy node

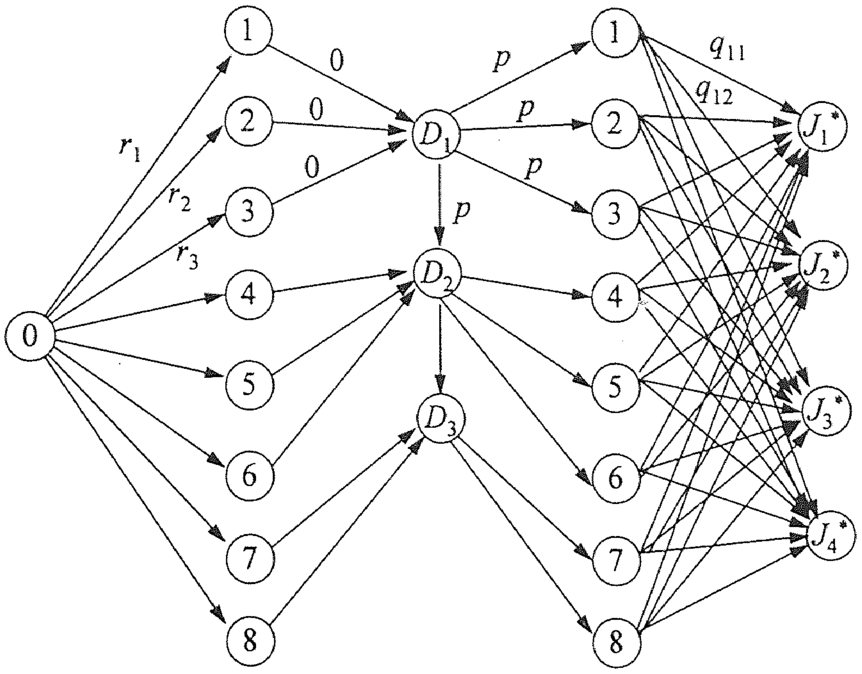


Figure 6-3: Graphical representation

6.3 Problem Analysis

When m operations are processed on the same batch, the release time of the batch is $\max\{r_1, \ldots, r_m\}$ and the processing time is $\max\{p_1,\ldots, p_m\} = p$. The starting time of the batch is its release time or the completion time of the prior batch, whichever comes later. In the sequencing graph, the batch is represented by the dummy node. The job completion time is the earliest starting time of the associated sink. As the arc connecting the nodes to the dummy node has weight zero, the starting time of the batch is the maximum of the operation release time and the machine available time from the previous batch.

Proposition 1: For the batch machine scheduling problem with agreeable r_i and q_i and single job (n=1), there is an optimal sequence that follows batch longest tail order.

Proof: Agreeable r_i and q_i denote that the higher q_i has the lower r_i. Suppose that there exists the optimal sequence that do not follow the batch-longest tail order. Operation i is sequenced in batch b and operation i' is sequence in batch b'. Batch b' follows batch b, and $q_{i'} \geq q_i$ and $r_{i'} \leq r_i$. We will show that interchanging i and i' can decrease the job completion time which will reduce the objective value.

First, let us consider the case where $r_{i'} = r$.

$$C(i \rightarrow i') = \max\left\{\begin{matrix} \cdots \\ t^* + p + \max\left(q_1^b, \ldots, q_i, \ldots, q_{n_1}^b\right) \\ t^* + 2p + \max\left(q_1^{b+1}, \ldots, q_{n_1}^{b+1}\right) \\ \cdots \\ t^* + (b' - b)p + \max\left(q_1^{b'}, \ldots, q_{i'}, \ldots, q_{n_1}^{b'}\right) \\ \cdots \end{matrix}\right\},$$

$$C(i' \rightarrow i) = \max\left\{\begin{matrix} \cdots \\ t^* + p + \max\left(q_1^b, \ldots, q_{i'}, \ldots, q_{n_1}^b\right) \\ t^* + 2p + \max\left(q_1^{b+1}, \ldots, q_{n_1}^{b+1}\right) \\ \cdots \\ t^* + (b' - b)p + \max\left(q_1^{b'}, \ldots, q_i, \ldots, q_{n_1}^{b'}\right) \\ \cdots \end{matrix}\right\},$$

where,

$C(i \rightarrow i')$	is the job completion time when operation i precede i',
q_i^b	is the tail of operation i in batch b,
t^*	is the starting time of operation i in the sequence that $i \rightarrow j$.

Because $q_{i'} \geq q_i$ and $q_{\hat{i}}^b \geq q_{\tilde{i}}^{b'}$ where $\hat{i} = 1, \ldots, i-1, i+1, \ldots, n_b$ and $\tilde{i} = 1, \ldots, i'-1, i'+1, \ldots, n_{b'}$,

$t^* + (b' - b)p + \max\left(q_1^{b'}, \ldots, q_i, \ldots, q_{n_b}^{b'}\right) \leq t^* + (b' - b)p + \max\left(q_1^{b'}, \ldots, q_{i'}, \ldots, q_{n_b}^{b'}\right)$ and

$$t^* + (b'-b)p + \max\left(q_1^{b'},\ldots,q_i,\ldots,q_{n_b}^{b'}\right) \le t^* + p + \max\left(q_1^{b},\ldots,q_{i'},\ldots,q_{n_b}^{b'}\right).$$

Therefore, C($i \rightarrow i'$) ≤ C($i' \rightarrow i$).

Now, consider the case that r_i increases as q_i decreases. The starting time of batch b will not increase after the interchange as $r_{i'} \le r_i$. Therefore, C($i \rightarrow i'$) ≤ C($i' \rightarrow i$).

6.3.1 BEDD-O (Batch Earliest Operation Due Date Rule)

When $r_i = r$ and n>1, the problem becomes NP-hard as it can be reduced to a single machine scheduling problem which is proved to be NP-hard. Therefore, we modify the EDD-O rule for a new environment. The major difference is that, instead of one operation, z operations with the lowest indices will be selected before updating the machine release time on each step. BEDD-O always generates a full-batch sequence. The machine will start the process when there are z tasks waiting.

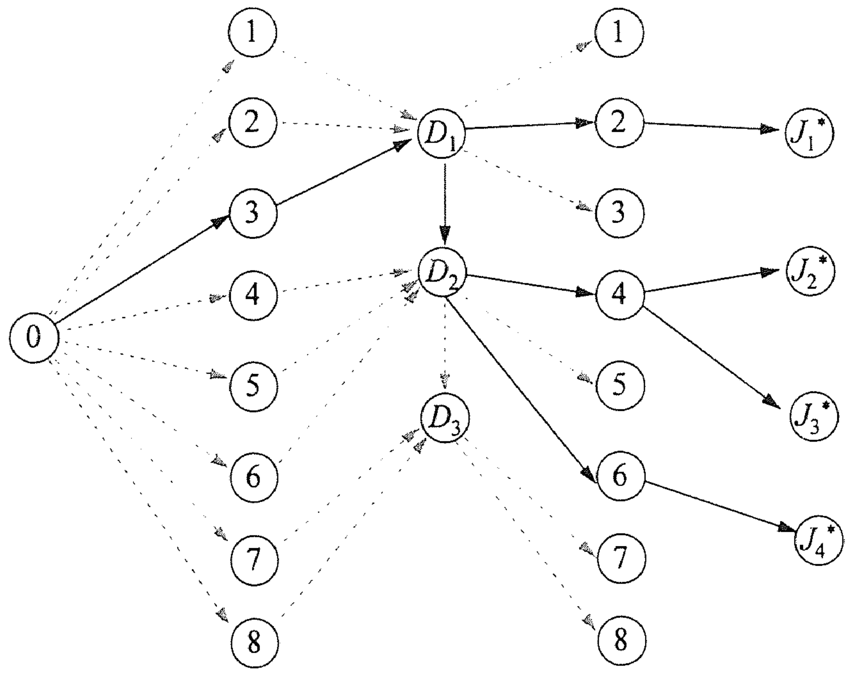


Figure 6-4: Critical paths on a sequenced graph

Proposition 2: The batch starting time can be reduced only by moving the critical operation to the following batch.

Proof: The starting time of the batches depends on the starting time of the first batch in the cluster. Therefore, only the critical operations on the first batch effect the batch starting time. Figure 6-5 is a partial graph of the first batch in Figure 6-4. Operation 3 is the critical operation of the batch. The completion time of job 1 can only be reduced if operation 3 is moved from that batch.

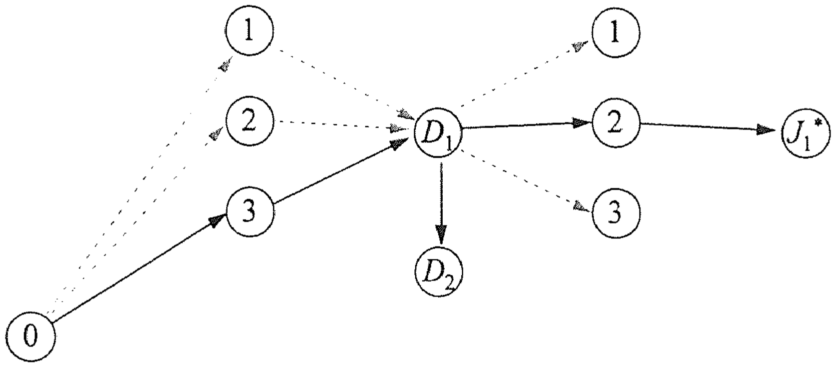


Figure 6-5: Starting time of the batch depends on operation 3

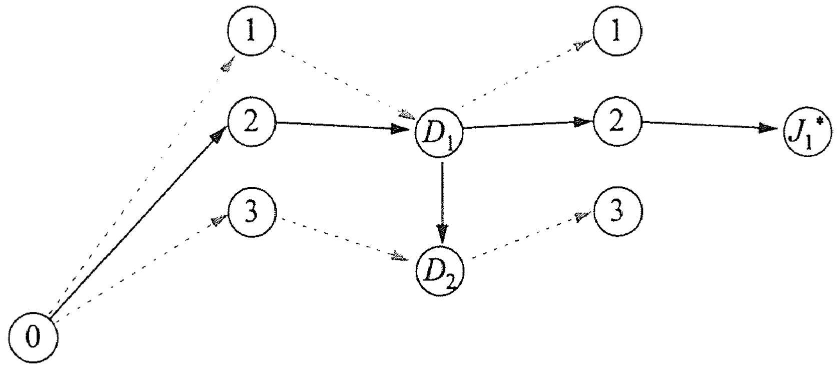


Figure 6-6: Reducing the batch starting time by move operation 3 to the next batch

Definition: *Batch sequence* is the sequence on the batch machine that includes a task assignment. For example, $S = \{\{1,2,3\},\{4,5,6\}\}$ represents the sequence on the machine that has two batches. Task 1, 2, and 3 are assigned to the first batch while task 4, 5, and 6 are assigned to the second batch.

Definition: *Linear sequence* is a sequence representation where all elements in the sequences are lined up into a string. If operation i precedes operation i' in the linear sequence, i is processes before or at the same time as i'. For example, operation 6 cannot start before operation 5 in the linear sequence: $\{1, 2, 3, 4, 5, 6\}$. In other words, operation 6 can only be in the same batch as operation 5 otherwise it has to be in a later batch.

Converting Batch Sequence to Linear Sequence

The batch sequence can be converted to a linear sequence by removing the batch assignment. First, the tasks in each batch should be sorted by their release time. Then, the batch assignment will be released. For example, $S = \{\{1,2,3\},\{4,5,6\}\}$ and $r_{i'} \leq r_i$ for $i < i'$ will be transformed to $\{1,2,3,4,5,6\}$.

6.3.2 Batching Heuristic (Dynamic Programming)

This heuristic attempts to assign operations in the line sequence to the batch in a way such that the objective value $\sum_{k=1}^{J} w_k(T_k + \exp\{-E_k\})$ is minimal. It is based on dynamic programming. On the single job case (n=1), this heuristic will find the optimal solution. When there are more than one job, it cannot guarantee the optimality.

Let $c^j(i)$ denotes the completion time of job j when operations 1 to i have been sequenced. $c_b^j(i)$ denotes the completion time of job j when operations 1 to i have been sequenced and operation i is assigned to the batch of size b. $f(i)$ represents the minimum completion time for operations 1, …, i according to the line sequence S. $f_b(i)$ denotes $f(i)$ with the restriction that operation i must be assigned to the batch of size b.

Then,

$$c_b^j(i) = \max\left(c^j(i-b), e_{b,i'}(i) + q_{i'}^{(j)}\right) \qquad \text{for } i' = i - b + 1, \ldots, m,$$

$$= \text{BigM} \qquad \text{if } \sum_{v=i-b}^{i} \frac{1}{p_{(i-v),i}} > \delta \text{, where } \delta \text{ is a very small number and}$$

BigM is a large number.

$$e_{b,i'}(i) = f_b(i) \qquad \text{if } i\text{-}b < i' \leq i,$$

$$= \max\left(e_b(i-b), t_{b,i''}(i) + p_{i'',i'} + p_{i'}\right) \qquad \text{for } i'' = 1,\ldots,i'-1,$$

$$f_b(i) = \max\left\{f(i-b), r'_{i-b+1}, r'_{i-b+2}, \ldots, r'_i\right\} + \max\left\{p_{i-b+1}, p_{i-b+2}, \ldots, p_i\right\},$$

$c^j(i) = c_b^j(i)$, $f(j) = f_b(j)$ and $z(i) = b$.

where, i minimize $Objective = \sum_{j=1}^{J} w^{(j)}T^{(j)} - h^{(j)}E^{(j)}$ where $T^{(j)} = \max(c_b^j(i) - d^{(j)}, 0)$

$E^{(j)} = \max(d^{(j)} - c_b^j(i), 0)$ for $j = 1, \ldots, \min(i,z)$,

and $c^k(0) =$ minimum completion time of job k determined from the directed graph,

$$f(0) = 0,$$

$$f(i) = \min\left\{f_b(i)\right\},$$

$$r'_{\hat{i}}(i) = \max\left\{r_{\hat{i}}, t_{i'''}(i) + p_{i\hat{i}}\right\},$$

$$t_{i'''}(i) = f(i) - \max\left\{p_{i-z(i)+1}, \ldots, p_i\right\} \qquad \text{if } i\text{-}z(i) \leq i''' \leq i\text{-}1,$$

$$= t_{i'''}(i - z(i)) \qquad \text{otherwise,}$$

$$
\begin{aligned}
e_{i'''}(i) &= f(i) && \text{if } i - z(i) < i''' \le i\,, \\
&= e_{i'''}(i - z(i)) && \text{otherwise,} \\
e_{i'''}(0) &= c^{j}(0)\,.
\end{aligned}
$$

Please note that if there is no precedence relationship between operation i and i', set $p_{ii'}$ = -BigM. In the same way, let $q_i^{(j)}$ = -BigM if job j does not depend on operation i.

Example 6-1. Consider an instance of $1|r_i, p_i, B| \sum_{j=1}^{n}\left(w^{(j)}T^{(j)} - h^{(j)}E^{(j)}\right)$ with these data:

Table 6-1: Job information for Example 6-1

Job	$d^{(j)}$	$w^{(j)}$	$h^{(j)}$	$rc^{(j)}$
1	15	1	0.01	0
2	17	3	0.03	0
3	18	2	0.02	0

Table 6-2: Operation information for Example 6-1

Operation	r_i	p_i	$q_i^{(1)}$	$q_i^{(2)}$	$q_i^{(3)}$
1	0	3	6	8	5
2	2	4	4	-	5
3	3	3	2	7	3
4	5	5	3	-	-
5	6	4	-	2	4

Table 6-3: Dependent table ($p_{ii'}$) for Example 6-1

Operation	1	2	3	4	5
1	■			3	
2		■		4	3
3			■		
4				■	
5					■

$f(0)=0$,

$$c^1(0)=\max\begin{Bmatrix}0+3+6\\2+4+4\\3+3+2\\6+5+3\\6+4-BigM\end{Bmatrix}=14,$$

$$c^2(0)=\max\begin{Bmatrix}0+3+8\\2+4-BigM\\3+3+7\\6+5-BigM\\6+4+2\end{Bmatrix}=13,$$

$$c^3(0)=\max\begin{Bmatrix}0+3+5\\2+4+5\\3+3+3\\6+5-BigM\\6+4+4\end{Bmatrix}=14,$$

$r_1'=r_1=0$,

$f_1(1)=\max\{f(0),r_1'(1)\}+\max\{p_1\}=0+3=3$,

$c_1^1(1)=\max\{c^1(0),f_1(1)+\max(q_{11})\}=\max\{14,3+6\}=14$,

$c_1^2(1)=\max\{c^2(0),f_1(1)+\max(q_{12})\}=\max\{13,3+8\}=13$,

$c_1^3(1)=\max\{c^3(0),f_1(1)+\max(q_{13})\}=\max\{14,3+5\}=14$,

$z(1)=1$, therefore

$f(1)=3$,

$c^1(1)=14$, $c^2(1)=13$, $c^3(1)=14$,

$t_1(1)=f(1)-p_1=0$.

$r_1'(2) = r_1 = 0$,

$r_2'(2) = r_2 = 2$,

$f_1(2) = \max\{f(1), r_2'(2)\} + \max\{p_2\} = \max\{3,2\} + 4 = 7$,

$f_2(2) = \max\{f(0), r_1'(2), r_2'(2)\} + \max\{p_1, p_2\} = \max\{0,0,2\} + \max\{3,4\} = 6$,

$c_1^1(2) = 15, c_1^2(2) = 13, c_1^3(2) = 14$,

$c_2^1(2) = 14, c_2^2(2) = 14, c_2^3(2) = 14$,

$Objective_1 = -0.01(0) - 0.03(4) - 0.02(4) = -0.2$,

$Objective_2 = -0.01(1) - 0.03(3) - 0.02(4) = -0.18$,

Therefore, $z(2) = 1$.

$f(2) = 7$,

$c^1(2) = 13, c^2(2) = 13, c^3(2) = 14$.

$t_1(2) = t_1(1) = 0$,

$t_2(2) = f(2) - p_2 = 3$.

$r_2'(3) = r_2 = 2$,

$r_3'(3) = r_3 = 3$,

$f_1(3) = \max\{f(2), r_3'\} + \max\{p_3\} = \max\{7,3\} + 3 = 10$,

$f_2(3) = \max\{f(1), r_2'(3), r_3'(3)\} + \max\{p_2, p_3\} = \max\{3,2,3\} + \max\{4,3\} = 7$,

$c_1^1(3) = 15, c_1^2(3) = 17, c_1^3(3) = 14$,

$c_2^1(3) = 15, c_2^2(3) = 14, c_2^3(3) = 14$,

$Objective_1$ = -0.08, $Objective_2$ = -0.17,

Therefore, $z(3) = 2$.

$f(3) = 7$, $c^1(3) = 15$, $c^2(3) = 14$, $c^3(3) = 14$.

$t_1(3) = 0$, $t_2(3) = 3$, $t_3(3) = 3$.

$f_1(4) = 12, f_2(4) = 12$,

$c_1^1(4) = 15, c_1^2(4) = 14, c_1^3(4) = 14$,

$c_2^1(4) = 15, c_2^2(4) = 19, c_2^3(4) = 15$,

$Objective_1$ = -0.17, $Objective_2$ = 5.94,

Therefore, $z(4) = 1$.

$f(4) = 12$, $c^1(4) = 15$, $c^2(4) = 14$, $c^3(4) = 14$,

$t_1(4) = 0$, $t_2(4) = 3$, $t_3(4) = 7$, $t_4(4) = 7$.

$f_1(5) = 16, f_2(5) = 12$,

$c_1^1(5) = 15, c_1^2(5) = 18, c_1^3(5) = 20$,

$c_2^1(5) = 15, c_2^2(5) = 14, c_2^3(5) = 16$,

$Objective_1$ = 7, $Objective_2$ = -0.13,

Therefore, $z(5) = 2$.

$f(5) = 12$, $c^1(5) = 15$, $c^2(5) = 14$, $c^3(5) = 16$,

$t_1(5) = 0$, $t_2(5) = 3$, $t_3(5) = 3$, $t_4(5) = 7$, $t_5(5) = 7$.

In summary, the sequence from batching heuristic is {{1},{2,3},{4,5}} which provides the objective value of -0.13.

6.4 Single Family

We have discussed a couple of techniques for batch machine scheduling. In this section, we combine these techniques with the single machine scheduling heuristic discussed in Chapter 4. The problem concerns only single family tasks with p_j=p and $s_{ii'} = 0$.

BEDD-O + Batching + Single Machine Scheduling (EBS)

To improve the sequence generated by BEDD-O, we transform the sequence to a linear sequence. Then, apply batching heuristic to the sequence. The batching heuristic will generate the best possible batching strategy. The idea behind it is that the sequence generated by BEDD-O is a full-batch. Imagine if a batch need to wait for a very long time for the next operation to start the processing. It might be better off to start the processing without that delayed operation.

After batches are assigned, we can look at them as new operations. The release time and the process time of these operations are the maximum release time and maximum processing time of operations in that batch. The sequencing technique developed for single machine scheduling can be applied.

According to the Proposition 2, the current sequence can be improved further by re-batching. The sequence will be transformed to a linear sequence and apply batching heuristic.

6.5 Multiple Families with Dependent Setup and Different Processing Time

Similar to the single family case, there are m tasks to be processed on the batch machine. However, these tasks are separated into f families. The tasks of different families cannot be processed on the same batch. Furthermore, setup is required when the machine switches from processing task of one family to another, and the processing time of the tasks may vary. The batch processing time, p^b, is equal to $\max_i(p_i)$ when operation i is in batch b.

Multiple Families Batching (MFB)

We extend the EBS heuristic for single family batch machine scheduling problem to the multiple family batch machine scheduling problem. The priority rule needs a modification to prevent operation from different families to be processed together. The priority rule determined the operation that has the highest (or lowest) index. Then it search for operations of the same family that will be put in the same batch.

There are three priority rules that provide good sequences – FCFS, EDD-O and modified EDD-O. Therefore, we select the best sequence from these rule as the starting sequence. Then, we apply Batching heuristic developed in the previous section to the sequence. There is a minor modification on the Batching heuristic to prevent operations from different families to be in the same batch. Then, CPI technique is applied to result. The processing to the batch need to be determined due to the fact that p_i may not equal to $p_{i'}$.

6.6 Numerical Examples

We create 50 batch machine scheduling problems with tails from decomposing the assembly shop problems. Table 6-4 to Table 6-7 compare the results of the heuristic with Full-Batch EDD which is used for minimizing maximum lateness by Lee *et al.*(1992), Full-Batch SPT, Full-Batch LPT and Full-Batch FCFS. We generate test problems by decomposing the assembly shop problems. The first 25 problems have 36 tasks while the rest have 39 tasks. EBS performs particularly well when jobs have tight due dates.

Then, we apply the similar technique to the problem with multiple families. Modified BEDD-O with batching and CPI provide good results. Most of the time, they provide better sequences than full-batch priority rules. The test problems are quite similar to the ones used previously. There are 36 operations divided into two families. The setup time ranges from 0 to 3 and the processing time range from 5 to 15.

The batching step does not provide very impressive improvement as in single family case. The objective value reduction of the batching and CPI over the best sequence generated by priority rules is around 2%.

Table 6-4: Objective values (single family BMSPT)

	SPT	LPT	FCFS	EDD-O	EBS
1	145.6	145.6	145.6	145.6	121.6
2	129.6	129.6	96.5	105.6	74.5
3	540.0	540.0	520.0	520.0	488.0
4	257.9	257.9	253.9	257.9	223.9
5	579.0	579.0	548.0	558.0	533.0
6	909.9	909.9	723.9	723.9	723.9
7	1197.8	1197.8	1197.8	1197.8	1189.8
8	1778.9	1778.9	1769.9	1763.9	1763.9
9	1672.9	1672.9	1672.8	1672.8	1672.8
10	1656.9	1656.9	1587.9	1647.9	1627.9
11	431.8	431.8	431.8	431.8	431.8
12	1099.9	1099.9	1099.9	1099.9	1095.9
13	1527.0	1527.0	1527.0	1597.0	1524.0
14	1744.9	1744.9	1744.9	1744.9	1580.9
15	1258.0	1258.0	1265.0	1258.0	1225.0
16	466.9	466.9	466.9	476.9	454.9
17	794.9	794.9	601.8	601.8	587.8
18	1042.8	1042.8	1042.8	1084.8	1000.8
19	1183.0	1183.0	1183.0	1183.0	1183.0
20	1139.9	1139.9	1139.9	1087.9	1068.9
21	11.7	11.7	-0.4	-0.4	-0.4
22	228.8	228.8	228.7	228.7	228.7
23	596.9	596.9	596.9	561.9	551.9
24	612.0	612.0	472.9	472.9	448.9
25	1550.9	1550.9	1517.9	1555.9	1451.9

Table 6-5: Objective values (single family BMSPT) (cont.)

	SPT	LPT	FCFS	EDD-O	EBS
26	932.8	932.8	206.7	336.8	282.8
27	1454.9	1454.9	1318.8	1370.9	1313.9
28	478.8	478.8	151.6	227.6	227.6
29	2052.0	2052.0	1784.0	1852.0	1830.0
30	1503.0	1503.0	1083.9	1187.9	1187.9
31	284.4	284.4	139.2	139.2	139.2
32	544.6	544.6	462.4	449.6	447.6
33	515.6	515.6	412.5	412.4	412.4
34	1051.9	1051.9	1030.8	998.8	977.8
35	1112.9	1112.9	280.6	258.7	258.7
36	209.6	209.6	63.6	63.6	63.6
37	478.7	478.7	272.6	226.7	223.7
38	142.6	142.6	13.5	13.5	13.4
39	1234.0	1234.0	1320.0	1121.0	1041.0
40	914.0	914.0	823.0	898.0	898.0
41	273.5	273.5	130.1	119.5	119.5
42	607.6	607.6	607.5	607.5	573.5
43	188.4	188.4	-1.1	-1.1	-1.1
44	884.8	884.8	869.8	864.8	854.8
45	725.8	725.8	725.8	725.8	658.8
46	192.7	192.7	77.5	77.5	77.5
47	1159.0	1159.0	1129.8	1129.9	1097.0
48	261.9	261.9	150.6	150.7	150.7
49	1741.0	1741.0	1721.0	1741.0	1723.0
50	954.9	954.9	915.9	922.9	905.9

Table 6-6: Variation from the minimum (single family BMSPT)

	SPT	LPT	FCFS	EDD-O	EBS
1	19.79%	19.79%	19.74%	19.78%	0.00%
2	74.00%	74.00%	29.60%	41.70%	0.00%
3	10.66%	10.66%	6.56%	6.56%	0.00%
4	15.19%	15.19%	13.40%	15.19%	0.00%
5	8.63%	8.63%	2.81%	4.69%	0.00%
6	25.69%	25.69%	0.00%	0.00%	0.00%
7	0.67%	0.67%	0.67%	0.67%	0.00%
8	0.85%	0.85%	0.34%	0.00%	0.00%
9	0.00%	0.00%	0.00%	0.00%	0.00%
10	4.35%	4.35%	0.00%	3.78%	2.52%
11	0.00%	0.00%	0.00%	0.00%	0.00%
12	0.36%	0.36%	0.36%	0.36%	0.00%
13	0.20%	0.20%	0.20%	4.79%	0.00%
14	10.37%	10.37%	10.37%	10.37%	0.00%
15	2.69%	2.69%	3.27%	2.69%	0.00%
16	2.64%	2.64%	2.63%	4.83%	0.00%
17	35.23%	35.23%	2.38%	2.38%	0.00%
18	4.20%	4.20%	4.20%	8.39%	0.00%
19	0.00%	0.00%	0.00%	0.00%	0.00%
20	6.64%	6.64%	6.64%	1.78%	0.00%
21	na.	na.	na.	na.	na.
22	0.07%	0.07%	0.01%	0.00%	0.00%
23	8.15%	8.15%	8.15%	1.81%	0.00%
24	36.32%	36.32%	5.35%	5.35%	0.00%
25	6.82%	6.82%	4.55%	7.16%	0.00%

Table 6-7: Variation from the minimum (single family BMSPT) (cont.)

	SPT	LPT	FCFS	EDD-O	EBS
26	351.33%	351.33%	0.00%	62.96%	36.81%
27	10.73%	10.73%	0.37%	4.33%	0.00%
28	215.72%	215.72%	0.00%	50.11%	50.11%
29	15.02%	15.02%	0.00%	3.81%	2.58%
30	38.66%	38.66%	0.00%	9.59%	9.59%
31	104.37%	104.37%	0.03%	0.02%	0.00%
32	21.67%	21.67%	3.31%	0.46%	0.00%
33	25.00%	25.00%	0.01%	0.00%	0.00%
34	7.57%	7.57%	5.41%	2.15%	0.00%
35	330.16%	330.16%	8.47%	0.00%	0.00%
36	229.69%	229.69%	0.00%	0.01%	0.01%
37	114.00%	114.00%	21.88%	1.33%	0.00%
38	961.70%	961.70%	0.35%	0.30%	0.00%
39	18.54%	18.54%	26.80%	7.68%	0.00%
40	11.06%	11.06%	0.00%	9.11%	9.11%
41	128.93%	128.93%	8.93%	0.00%	0.00%
42	5.94%	5.94%	5.92%	5.92%	0.00%
43	na.	na.	na.	na.	na.
44	3.51%	3.51%	1.76%	1.17%	0.00%
45	10.17%	10.17%	10.17%	10.17%	0.00%
46	148.77%	148.77%	0.09%	0.00%	0.00%
47	5.65%	5.65%	3.00%	3.00%	0.00%
48	73.83%	73.83%	0.00%	0.04%	0.04%
49	1.16%	1.16%	0.00%	1.16%	0.12%
50	5.41%	5.41%	1.10%	1.88%	0.00%
Avg	64.84%	64.84%	4.56%	6.62%	2.31%

Table 6-8: Objective value (multiple families BMSPT)

Problem	ACT (1,1)	SPT	LPT	FCFS	EDD-O	mod. EDD-O	(A-1)	(A-2)	(A-3)	reduc.
1	7542	1181	1600	736	1117	979	736	736	736	0%
2	6539	5418	2598	2412	3076	2284	2284	2284	2284	0%
3	6647	1341	1249	775	773	1109	773	773	773	0%
4	9422	4369	4063	3933	3925	3484	3484	3484	3484	0%
5	7470	2845	2492	2496	2852	2232	2232	2232	2232	0%
6	4337	984	1921	745	526	616	526	526	526	0%
7	5520	2039	3906	1907	2065	1610	1610	1610	1610	0%
8	3946	1150	1645	1356	839	1322	839	839	839	0%
9	6490	4052	2603	2178	1868	1548	1548	1548	1548	0%
10	5375	3375	1921	1755	1435	1591	1435	1435	1435	0%
11	6559	1968	702	1353	1305	249	249	249	249	0%
12	9181	2243	2387	2546	2467	1115	1115	985	985	12%
13	6965	2395	582	283	1206	147	147	123	123	16%
14	10832	4219	4668	4831	4670	3069	3069	3069	2980	3%
15	6985	5238	2565	3511	3479	2432	2432	2342	2293	6%
16	2203	941	487	290	508	530	290	290	290	0%
17	6225	1983	1592	1144	1055	806	806	806	806	0%
18	2846	1358	551	320	538	480	320	320	320	0%
19	6161	2276	2817	2267	2177	2032	2032	1856	1856	9%
20	5213	1986	1992	1885	1988	1564	1564	1564	1564	0%
21	4155	1041	612	363	603	816	363	363	363	0%
22	7262	3208	3161	3080	2466	2353	2353	2353	2353	0%
23	4769	1248	1312	1451	611	914	611	611	611	0%
24	7376	3764	3237	3632	3180	2534	2534	2534	2534	0%
25	7640	2365	1682	2266	1806	1693	1693	1595	1595	5%

Note: (A-1) : Best sequence generated by FCFS, EDD-O and modified EDD-O

(A-2) : (A-1) and Batching

(A-3) : (A-2) and CPI

CHAPTER 7

ASSEMBLY SHOP SCHEDULING BASED ON SHIFTING BOTTLENECK

In this chapter, we describe the method to sequence machines in the assembly shop. The objective that we are focusing on is to minimize the weighted tardiness. First, we explain the techniques to modify standard priority rules so that they can be used for the assembly shop scheduling problems. These rules are used as a comparison basis. Then, we discuss the heuristic for assembly shop scheduling with single machine workstations, which is based on shifting bottleneck concept (a decomposition technique). It is extended to consider the sequence dependent setup and multiple types of workstations. The results from chapters 3, 4, 5 and 6 are required in order to develop the heuristic.

7.1 Priority Rules

There are several priority rules that can be applied to the problem with little modifications. Many standard rules including Earliness Due Date (EDD), Shortest Processing Time (SPT), Longest Processing Time (LPT) are developed for standard single machine scheduling problems. These rules sort the jobs according to their indexing functions. The job with the highest index is selected as the next job to be processed. The steps are done iteratively until all jobs are sequenced.

For the assembly shop, jobs need to be processed by a series of machines. Each processing, called operation, can be considered as the job in the prior sense. The operations are assigned to the machines. When applying a priority rule, it might generate an infeasible solution if we do not consider the ongoing partial sequence. The assembly operation cannot start until all its prior operations have been completed. In order to apply the standard priority rules to the assembly shop problem, the following issues need to be considered:

(i) operation dependency

(ii) machine assignment

(iii) batch assignment.

7.1.1 Operation Dependency

A job in the assembly shop needs to visit various workstations according to its predefined route. The order of the visit cannot be altered. For example, a steal tube need to be cut before bent. It cannot be done backward. This creates the precedent relationship between operations. We add an investigation step to check for this dependency to prevent the sequence *dead-lock*. The dead-lock occurs when there is a cycle in the selection. For example, sequencing operation 1 after operation 4 on machine A and operation 3 after operation 2 on machine B will create a lock up. Operations 1 to 4 can never be started (Figure 7-1).

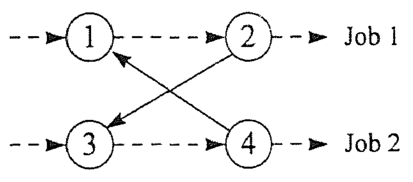


Figure 7-1: Sequence lock-up

The Method

The method is based on the directed graph, $D = \{N,C\}$. First, remove conjunctive arcs $(0,j)$ for all $j \in N$ from D. Node j without arc (i,j) for all $i \in N$ is defined as *active node*. Let A be a set of active nodes. We apply the priority rule on set A. Index all the nodes in A and select node j that $Index(j) \geq Index(k)$ for all $k \in A$. Sequence j and remove (j,k) for all $k \in N$. Let $A = A + \{i;\ |(k,i)| = 0 \text{ for all } k \in N\} - \{j;\ |(j,k)| = 0 \text{ for all } k \in N\}$. Repeat the steps until all nodes are sequenced. This method will guarantee that lock-up will never occur.

7.1.2 Machine Assignment

For the parallel machine workstation, the operation can be processed by one of the machines in the workstation. Let M_i be the set of machines that is able to process operation i. When $|M_i| > 1$, the standard priority rules cannot judge which machine it should assign the operation to. We attach a simple machine assignment rule to the priority rules. It assigns the operation to the machine that is able to complete the task earliest. If all machines have the same speed, the operation is assigned to the first available machine. The operation is assigned to the fastest machine if all the machines are available.

7.1.3 Batch Assignment

The next deficiency of the standard priority rules is the lack of batching rule. When the machine can process multiple jobs in a single run, the standard priority rules cannot decide whether the machine should wait for the next task or should start the processing immediately. We apply a simple rule for the batch assignment. It is referred as *full-batch* assignment. The batch will be started when it is full or no more job of the same family arrives.

7.2 Assembly Shop with Single Machine Workstations

We have discussed the modifications of the priority rules that extend their capabilities to assembly shop scheduling. These priority rules are one pass heuristics. Most of them provides a reasonably good sequence in a flash. However, there exists needs for better heuristics. Shifting bottleneck (SB) heuristic based on machine decomposition concept has proved to provide good solutions for classical job-shop scheduling problems. We extend this concept to our problem. The objective is to minimize the weighted tardiness which is different from the original SB by Adams *et al.* (1988).

7.2.1 Disjunctive Graph

The disjunctive graph $G(N,A,E)$ can be developed from the problem. N is a set of nodes that corresponding to the set of operations. The delay precedent constraints in the job routes are represented by arcs in set A. E is a disjunctive arc set. The disjunctive arc pairs connect operations that are processed on the same workstation. The arcs lengths, both conjunctive and disjunctive arcs, are equal to the processing time of the operations that they are originated from. A source node, node 0, and n sink nodes are added to N. Conjunctive arcs connecting the source node to the first operations in job, j, with length $r^{(j)}$ are added to represent the job release time. The arc linking the last operation, i, in job j to sink node (j) with length p_i are appended to A. Assembly operations are represented by fork type links. See section 2.3 for further details.

7.2.2 Workstation Objective

The assembly shop scheduling problem can be decomposed to sub-problems using the technique discussed in Section 3.2. We may use the objective of the main problem, minimize the weighted tardiness, on the sub-problems. However, the deficiency of this objective occurs when there are several sequences that provide the same objective value. In other words, there are several sequences for the sub-problem that complete all the jobs before their due dates. The weighted tardiness of these sequences will be zero. It is likely to occur when the shop has light to medium load. In this case, we will not be able to judge which sequence should be selected. The selection is done randomly.

We enhance the heuristic by modifying the objective of the sub-problems. The new objective will have two goals. The first goal is to minimize the weighted tardiness. The next goal is to maximize the weighted earliness. This problem can be formulated as a goal programming. The objective can be written as lexmin u = { $\min\sum_j w^{(j)}T^{(j)}$; $\max\sum_j w^{(j)}E^{(j)}$ } where $w^{(j)}$ is the priority of job j, $T^{(j)}$ is the job tardiness; $T^{(j)} = \max(C^{(j)} - d^{(j)},0)$, $E^{(j)}$ is the job earliness; $E^{(j)} = \max(d^{(j)} - C^{(j)},0)$. First, we consider the first goal, the weighted tardiness. If there is a number of sequences that can complete the jobs before their due dates, we select the sequence that gives the highest weighted earliness. Otherwise, select the sequence that provides the minimum weighted tardiness. The method breaks the tight by selecting the sequence that provides the highest slacks.

This goal programming can be replaced by a new single objective, $\min\sum_j\left(w^{(j)}T^{(j)} - h^{(j)}E^{(j)}\right)$; where $h^{(j)}$ is the earliness penalty of job j. When $w^{(j)} >> h^{(j)}$, this objective would be similar to minimizing the weighted tardiness with a special property. It will maximize the weighted earliness when no job is tardy. Due to the fact that the earliness penalty of the job is the reverse of the tardiness weight (because of the minus sign), the job with highest tardiness weight should have the lowest earliness penalty. Therefore, $h^{(j)}$ may be assigned with weight $\frac{1}{c \cdot w^{(j)}}$ where c is separating parameter. The separating parameter, c, provides the gap between the weighted tardiness and the weighted earliness. The weighted earliness will be at least c times below the weighted tardiness. When c is high, the weighted tardiness will dominate the weighted earliness. The objective will be $\min\sum_j w^{(j)}T^{(j)}$ when $h^{(j)}$=0 or c is infinite. In other words, minimizing weighted tardiness is a subset of this objective. Please note that to avoid divided by zero error, the job (tardiness) weights should be set positive and greater than zero.

7.2.3 Problem Decomposition

SB is based on the intuition that highly utilized machine should be sequenced first. The less utilized machines have longer slacks. Therefore, there is higher chance to find an available time to process an operation on those machines. The heuristic is done iteratively. First, find the most bottleneck machine among the unscheduled machines.

Then, determine the sequence of that machine and fix the selection on the directed graph. The steps are repeated until all the machines are sequenced.

Base on the disjunctive graph, G, we decompose the problem into sub-problems according to the machine (clique). The method has been proposed in section 3.3. The nodes that are not related to the machine are removed. The final graph contains the source node, the operations that will be processed on the machine, the sinks, and links. The objective of the sub-problem is to find the sequence that minimize the weighted tardiness-earliness. This problem has been discussed in details in chapter 4.

7.2.4 Bottleneck Determination

We define the bottleneck machine as a machine that provides the highest objective value on the decomposed problem. It is the machine that has high potential to delay high priority tasks. Solving the decomposed problem to the optimal or near optimal may take a considerable amount of time; therefore, we apply a priority rule, TER, to the problem. This rule provides a rough estimation which is good enough for the heuristic.

7.2.5 Sequencing the Bottleneck

After the bottleneck machine is identified, we determine the sequence of that machine. We apply the result from chapter 4 on this problem. The critical path improvement method (CPI) is applied to the sequence determined by TER.

7.2.6 Updating the Graph

After the sequence of the bottleneck is determined, the selection according to the sequence is added to the graph (Figure 7-2). We use the modified selection method in the heuristic. As some of the arcs in the selection are redundant, they can be discarded. For example, arc (4,6),(4,2), and (10,2) in Figure 7-2 are redundant. They can be discarded. The modified selection can be replaced by the one in Figure 7-3.

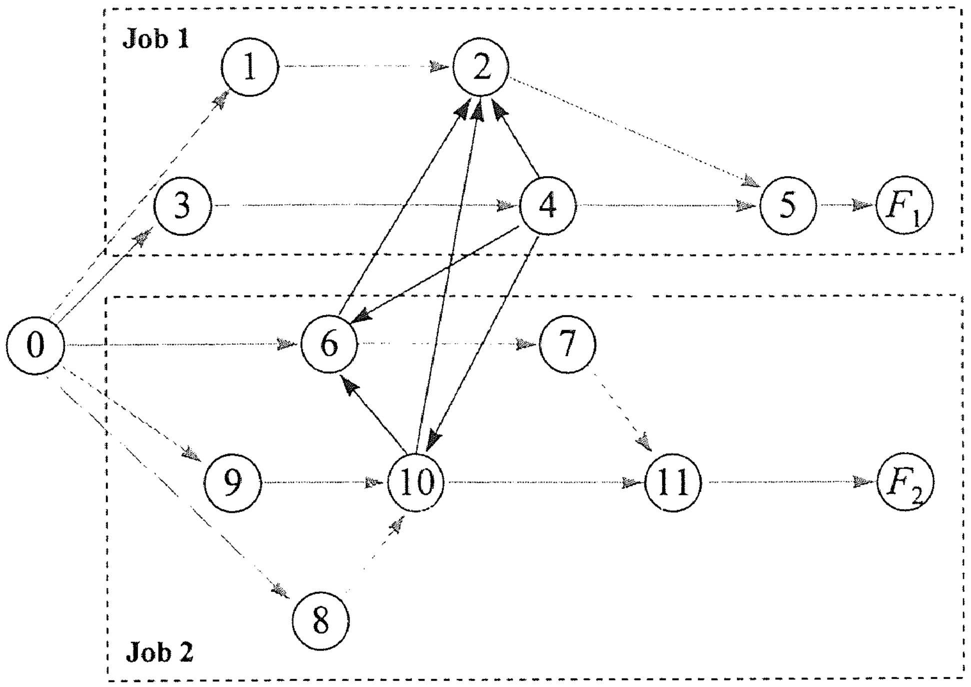


Figure 7-2: Adding a selection on machine 2

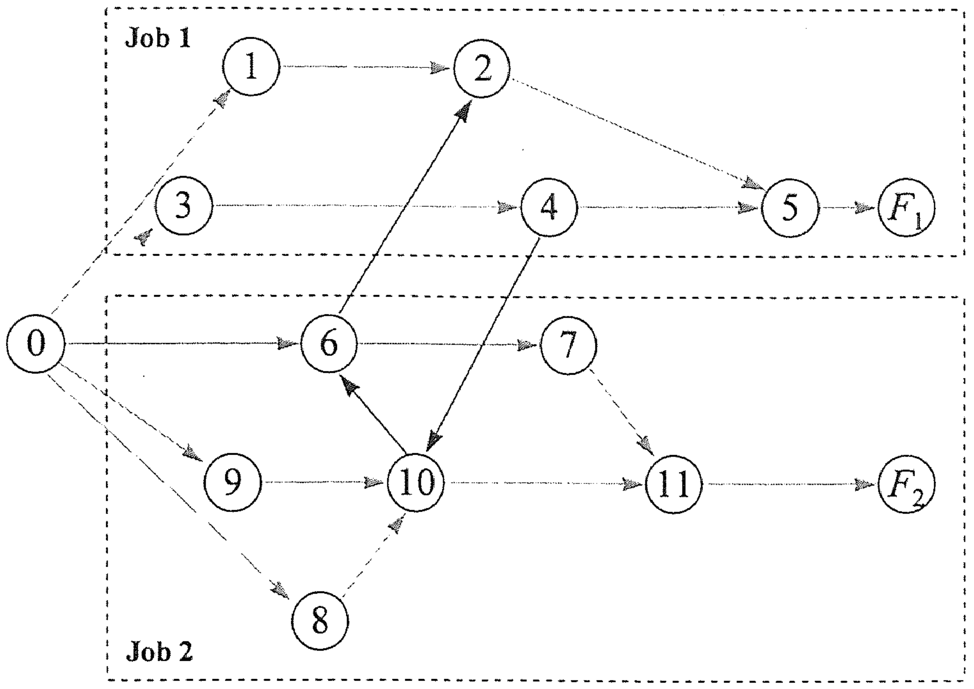


Figure 7-3: The modified selection

The arc (i,j) in the selection has length p_i. If the machine available time is considered, arc $(0,l)$ where l is the first operation in the sequence will be added with length r_m.

7.2.7 Local Re-Optimization

This step is added to improve the sequences of the machines in S. It can be perceived as a local search method. Instead of randomly swapping the operations in the sequence, we smartly determine a better sequence on the previously sequenced machine if it exists.

We remove m (=4) most recent sequences of the machines from the graph one after another. The new sub-problem is reconstructed after each removal. This problem is different from the one previously solved because some new links have been added in the previous steps. We solve this problem by using the similar technique in determining the sequence for the bottleneck machine. We hope that re-solving the problem will generate a better sequence. The sequence is added to the graph before removing the next sequence.

7.2.8 Global Optimization

This step is the last step of the heuristic. It is similar to local re-optimization. The difference is that this step is done extensively. We remove all the sequences of the machines and replace them with better sequences one after another. The step guides the search to a new local minimum.

Example 7-1 We use the example in chapter 3 to demonstrate the heuristic. We assume that all workstations are single machine workstations.

Iteration 1

Unscheduled machines: 1, 2, 3

Scheduled machines: ∅

The problem is decomposed to three sub-problems (# of unscheduled machines). By applying TER to the sub-problems, we obtain three sequences. The sequence on machine 1 {M1: 2, 6, 8, 1, 11, 13} has objective value = 24 while {M2: 7, 4, 10, 12, 15} and {M3: 3, 9, 5, 14} have the values of -0.07 and 8 respectively. Therefore, machine 1 is concerned as the bottleneck machine.

Bottleneck machine: 1 (determined by TER) We, then, apply CPI to the sequence. The result is {M1: 6, 8, 2, 1, 11, 13} which has objective value of 20.

Iteration 2

Unscheduled machines: 2, 3

Scheduled machines: 1

Similar to the previous iteration, the problem is decomposed to two sub-problems. The bottleneck machine is the machine that its sequence has the highest objective value as determined by TER.

Bottleneck machine: 3 (determined by TER) The objective value is 22 and the sequence is {M3: 9, 3, 5, 14}. After applying CPI, there is no improvement.

Local re-optimization step is performed by removing the sequence on machine 1, the previously scheduled machine. The new sub-problem on machine 1 is formulated. It is solved by TER+CPI. The result is appended back. However, we do not find the improvement.

Sequence summary: {M1: 6, 8, 2, 1, 11, 13; M3: 9, 3, 5, 14}

Iteration 3

Unscheduled machines: 2

Scheduled machines: 1, 3

Bottleneck machine: 2 (determined by TER) The objective value is 22 and the sequence is {M2: 7, 4, 10, 12, 15}. As the objective value does not increase from the previous step, we know that there is no better sequence. Therefore CPI is skipped.

Local re-optimization step is performed by removing the sequence on machine 3 and 1 respectively. The new sub-problems are formulated after each removal. The results from solving the sub-problems are appended back. However, we do not find the improvement.

Sequence summary: {M2: 7, 4, 10, 12, 15; M3: 9, 3, 5, 14; M1: 6, 8, 2, 1, 11, 13}

Final Re-Optimization

As the number of workstations is less than four, the final re-optimization is similar to the local re-optimization. We do not find further improvements.

Sequence summary: {M2: 7, 4, 10, 12, 15; M3: 9, 3, 5, 14; M1: 6, 8, 2, 1, 11, 13}

7.3 Assembly Shop with Single Machine Workstations and Dependent Setup Time

In a more complex problem where dependent setup time is considered, the structure of SB is generally the same as the previous one. There are two modifications on the heuristic. The first one is on the heuristic used to solve the sub-problems. The other is the graph updating step.

7.3.1 Sequencing the Bottleneck

Because the structure of the problem is altered, the sequencing method used in this step needs an improvement. The steps can be separated into two phases. In the first phase, we applied TER and CPI to generate a sequence which is similar to SB without dependent setup time. It is followed by 70 random search steps explained in chapter 4.

7.3.2 Updating the Graph

When adding the selection to the graph, the length of the arcs cannot be set to the operation processing time as they used to be. The arc (j,k) in the selection should have length equal to $s_{ij} + p_j$ where i is the prior operation to j in the sequence. This is equal to the time required for the machine to complete the task.

7.4 Assembly Shop with Multiple Types of Workstations

The structure of the heuristic for solving assembly shop scheduling problem with multiple types of workstations is similar to the ones with single machine workstations. The major difference is on solving the sub-problems. Although the objective for the sub-problems and job information are the same, the machine structure is difference. Jobs can be processed by one of the available machines in the parallel machine workstations. The batch machine can process a number of operations in a single run.

7.4.1 Problem Decomposition

The assembly shop will be decomposed to sub-problems. The sub-problems may be the single machine scheduling problem with tails, parallel scheduling problem with tails, or batch machine scheduling problem with tails. We utilize the results from Chapters 4, 5, and 6 on solving the sub-problems. We summarize the techniques used to solve each sub-problem when setup time is not concerned in the below.

(i) Single machine workstation: TER + CPI

First, the sequence is determined by TER, a priority rule. It provides a good starting point for CPI. By analyzing the critical paths on the scheduling graph, we can improve the sequence by moving some operations.

(ii) Parallel machine workstation: Beam search with beam width 3

Beam search is a good sequencing technique for parallel machine scheduling problems. Here, we apply PEDD-O as the evaluating function.

(iii) Batch machine workstation: BEDD-O + Batching + CPI

First, the sequence is generated by BEDD-O. A full batch sequence is generated. This sequence will be improved by Batching heuristic which will optimize the batch sizes. After the appropriate batch sizes are determined, each batch is perceived as a dummy operation. CPI will be applied to improve the sequence.

7.4.2 Bottleneck Determination

To improve the computational time, we use priority rules and a simple heuristic to approximate the indices on determining the bottleneck workstation. Because we have various types of workstations, different techniques will be used. The following list provide a summary of the techniques.

(i) Single machine workstation : TER

(ii) Parallel machine workstation : PEDD-O

(iii) Batch machine workstation : BEDD-O + Batching

7.4.3 Updating the Graph

After the sub-problem is solved, the sequence will be added to the disjunctive graph. The modified disjunctive graph discussed in Section 2.6.2 is used. The followings are the techniques to add the sequences for parallel machine workstations and batch machine workstations to the graph.

(i) *n* - Parallel machine workstation: *n* selections will be added to the clique associating with the workstation.

(ii) Batch machine workstation: Dummy nodes are added to represent the batch starting times and processing times. There are arcs linking the operations, say *i*, in the same batch to a dummy node and arcs linking this dummy node to the operations those have delay precedent constraints with operation *i*.

Example 7-2 We use the example in chapter 3 to demonstrate the heuristic. In the problem, there are three jobs (15 operations) to be processed on three workstations. The first workstation is "Cutting" that has two identical machines working in parallel. The second workstation is "Pressing" which has a single batch machine. This press machine can process up to two jobs in a single run. The last workstation is the assembly station. It is considered as a single machine workstation.

Iteration 1

Unscheduled workstation: 1, 2, 3

Scheduled workstations: ∅

The problem is decomposed to three sub-problems (# of unscheduled workstations). By applying TER to single machine workstation, PEDD-O to parallel machine workstation and BEDD-O + Batching to batch machine workstation, we obtain three sequences. The sequence on workstation 1 {M1: 2, 1, 11; M4: 6, 8, 13} has objective value = 1.98 while the sequence on workstation 2 and 3 are {M2: 7, 0, 4, 0, 12, 0, 10, 0, 15} and {M3: 3, 9, 5, 14} having the objective values of -0.11 and 8 respectively. Therefore, workstation 3 is considered as the bottleneck workstation.

Bottleneck workstation: 3 (determined by TER) We applied CPI to the sequence. The result is {M3: 9, 14, 3, 5} which has objective value of 7.92.

Iteration 2

Unscheduled workstations: 1, 2

Scheduled workstations: 3

Similar to the previous iteration, the problem is decomposed to two sub-problems. The bottleneck machine is the machine that its sequence determined by PEDD-O (parallel machines workstation) or BEDD-O+Batching (batch machine workstation) has the highest objective value.

Bottleneck workstation: 1 (determined by PEDD-O) The objective value is 12.96 and the sequence is {M1: 2, 11, 1; M4: 6, 8, 13}. After applying Beam search, the objective value is reduced to 9.94, and the new sequence is {M1: 6, 13, 1; M4: 8, 11, 2}.

Local re-optimization step is performed by removing the sequence on workstation 3, the previously scheduled workstation. The new sub-problem on machine 1 is formulated. It is solved by TER + CPI. The result is appended back. However, we do not find the improvement.

Local reopt! Before obj = 9.94

Sequence summary: {M1: 6, 13, 1; M4: 8, 11, 2}, {M3: 9, 14, 3, 5}

Iteration 3

Unscheduled workstations: 2

Scheduled workstations: 3, 1

Bottleneck workstation: 2 (determined by BEDD-O + Batching) The objective value is 9.94 and the sequence is {M2: 7, 0, 12, 4, 10, 0, 15, 0}. As the objective value does not increase from the previous step, we know that there is no better sequence. Therefore CPI is skipped.

Local optimization step is performed by removing the sequence on workstation 1 and 3 respectively. The new sub-problems are formulated after each removal. The results from solving the sub-problems are appended back. However, we do not find the improvement.

Sequence summary: {M1: 6, 13, 1; M4: 8, 11, 2}, {M3: 9, 14, 3, 5}, {M2: 7, 0, 12, 4, 10, 0, 15, 0}

Final Re-Optimization

As the number of workstations is less than four, the final re-optimization is similar to the local re-optimization. We do not find further improvements.

Sequence summary: {M1: 6, 13, 1; M4: 8, 11, 2}, {M3: 9, 14, 3, 5}, {M2: 7, 0, 12, 4, 10, 0, 15, 0}

7.4.4 Dependent Setup

As we assume that the sequence dependent setup time is small comparing to the processing time, there are minor modifications on the heuristic. The steps on SB remains unchanged. The modifications are on heuristics used in solving the decomposed problems. However, the heuristics used to determine bottleneck workstation remain unchanged. The modified heuristics for the sub-problems are listed below.

(i) Single machine workstation : TER ⇨ CPI ⇨ Local search(70)

(ii) Parallel machine workstation : Beam Search with beam width 3

(iii) Batch machine workstation : Three batch priority rules ⇨ Batching ⇨ CPI

7.5 Numerical Examples

We test a number of priority rules with the proposing heuristic. Each rule is briefly explained below.

SPT-1: This rule is a modification of SPT rule (for single machine) on assembly shop. The late arrival jobs are not allowed to be sequenced until the time has increased to some point. There exists at least an operation that can be scheduled before these late arrival jobs and it can complete the processing before their release time.

SPT-2: We prevent the generation of dead locks in the sequence by prohibiting non-active operations to be selected as the next operation to scheduled. The active operation is the operation that does not have active precedent constraint (no precedent constraint). Among the active operations, the one with shortest processing time is selected

LPT: Similar to SPT-2, this rule select the active operation that has the longest processing time.

EDD-O: The rule determines local due date for each operation by backward assignment. It select the operation with earliest due date for the next processing.

Modified EDD-O: Similar to SPT-1, modified EDD-O do not allow late arrival jobs to be sequenced until the time has increased to some point. The operations that are able to be sequenced will be selected using operation due date information.

EDD-J: This rule selects the operation that belongs to the job with minimum due date to process next.

FCFS: This rule assigns the processing sequence according to the time the job arrives at the workstation.

ATC: The Apparent Tardiness Cost rule selects the active operation according to an index. The index function is defined as $I_j(t) = \frac{w_j}{p_j}\exp\left(-\frac{\max(d_j - p_j - t, 0)}{k\bar{p}}\right)$ where t is the time at which the machine became free k is the scaling parameter and $\bar{p}$ is the average processing times of the remaining jobs.

SB w/o re-opt: This version of Shifting bottleneck does not include the local and global re-optimizations.

SB /w re-opt: It is the regular Shifting bottleneck for weighted tardiness-earliness.

We test the heuristic on Pentium 200 MHz PC running Microsoft Windows. We did not spent efforts on optimizing the program. Most of the codes are designed for general purpose scheduling heuristic development. The computation time can be reduced further by improving the codes. Forty tested problems are generated. The problems are separated into 2 groups – light load and high load. For the light load problems, the processing times of the operations are randomly assigned between 3 to 10. The job weights are between 1 to 5. The job due dates are assigned at 1.3 to 1.6 times the required processing times. For the high load problems, the processing times are between 5 to 15. The job weights are between 1 to 5. The due dates are assigned at 1.3 to 1.4 times the required processing times. The job release times are between 0 to 170 for both types of problems. We vary the problem sizes from 5 jobs with 10 operations each on 5 machines to 10 jobs with 15 operations each on 10 machines. The average number of assembly operations per job with 10 operations is 2.

Table 7-1 and Table 7-2 show the results for assembly shop with single machine workstations. The improvement of SB /w re-optimizations is compared with the best sequence generated by eight priority rules. SB provides approximately 36% improvement (objective reduction). However, there are some cases that SB generates poor sequences. By extending the global optimization step, better solution is expected. It is a trade-off between the quality of solution and computation time.

Table 7-3 shows the results when sequence dependent setup is considered. The setup time ranges from 0 to 15. We compare the heuristic with some standard priority rules. It is shown that SB can outperform priority rules in most cases. The average improvement over the minimum objective found is 19%. The computation time for the 10 jobs with 15 operations each on 10 machines is less than 200 seconds on Pentium 200 MHz.

Then, we modified the problems to include multiple types of workstations. Table 7-4 and Table 7-5 show the results for assembly type job-shop scheduling with various types of workstations. The SB can significantly outperform all priority rules that are tested. The similar results are noticed when sequence dependent setup is considered. Table 7-6 shows the numerical test results.

Table 7-1: Results from 5 machines assembly shop scheduling problems

Problem	# of oper.	SPT-1	SPT-2	LPT	EDD-O	modified EDD-O	EDD-J	FCFS	ATC(1)	SB w/o re-opt		SB /w re-opt		
										obj.	time (sec)	obj.	time (sec)	improv.
1	5x10	228	1596	1183	36	0	394	70	1352	9	1	0	4	0%
2	5x10	137	1369	1725	211	0	157	55	1383	100	1	68	5	-
3	5x10	746	2626	1068	368	174	220	101	1807	128	2	63	4	38%
4	5x10	194	2421	2275	151	0	702	41	1847	0	1	0	6	0%
5	5x10	477	1399	2429	148	118	281	287	624	19	1	16	6	86%
6	5x10	1160	2199	3246	586	465	947	707	2076	263	2	263	11	43%
7	5x10	655	3088	4975	547	305	916	485	2706	77	2	77	7	75%
8	5x10	1004	2590	3559	753	673	1473	701	2046	419	2	411	8	39%
9	5x10	1359	4036	3733	865	647	1175	858	2470	609	3	371	8	43%
10	5x10	992	2048	2777	975	750	1581	839	1636	336	2	336	11	55%
11	10x10	2485	7985	7732	1884	1481	3063	1833	5015	630	11	592	58	60%
12	10x10	1410	7228	9316	1111	638	865	917	5251	455	12	404	58	37%
13	10x10	1407	8207	9336	344	207	693	414	5593	603	12	260	48	-26%
14	10x10	559	7756	10272	598	230	1037	345	5446	211	9	211	41	8%
15	10x10	1171	8358	8088	1262	801	1476	992	6493	535	14	511	82	36%
16	10x10	3230	9345	9232	2585	1830	3825	2033	7374	2053	15	1349	73	26%
17	10x10	3879	11005	12803	2432	1585	4798	1987	6649	1187	14	1043	91	34%
18	10x10	4426	12773	12976	2353	1814	4471	2389	8040	1913	24	1727	112	5%
19	10x10	3960	9548	10663	1537	1296	3033	1895	4815	1168	25	753	118	42%
20	10x10	3557	8295	10666	2911	2312	4223	2331	6934	1825	20	1583	67	32%

Table 7-2: Results from 10 machines assembly shop scheduling problems

Problem	# of operations	SPT-1	SPT-2	LPT	EDD-O	modified EDD-O	EDD-J	FCFS	ATC(1)	SB w/o re-opt		SB /w re-opt		
										obj.	time (sec)	obj.	time (sec)	improv.
1	10x10	891	5639	4829	399	287	2195	429	3706	314	6	251	28	13%
2	10x10	1312	8021	5448	908	401	4072	531	4121	388	6	255	31	36%
3	10x10	439	4068	4439	325	259	1914	184	2213	46	6	46	24	75%
4	10x10	1250	5632	4520	1019	372	2882	402	3833	219	10	178	31	52%
5	10x10	485	4209	4008	517	315	2334	154	2070	161	5	144	27	6%
6	10x10	1451	4912	7887	1550	1097	3091	858	3000	477	6	307	28	64%
7	10x10	2320	6968	6246	2291	999	2974	877	3738	876	8	755	27	14%
8	10x10	2117	6663	8285	1400	1218	2293	1189	4267	683	8	604	39	49%
9	10x10	1992	6141	8620	2907	1542	4969	1553	4361	1216	11	785	47	49%
10	10x10	1773	5388	10279	2275	1332	4571	1573	4994	1203	16	1099	52	17%
11	10x15	533	6285	6835	407	331	2869	290	5022	543	12	138	39	52%
12	10x15	436	4384	4349	701	227	2246	270	3661	177	10	189	44	17%
13	10x15	570	6544	7122	416	169	2201	339	4472	245	12	95	91	44%
14	10x15	1251	8206	9188	1069	537	3206	818	5021	638	21	470	57	12%
15	10x15	776	8466	6910	707	313	2761	404	4641	320	16	63	130	80%
16	10x15	1052	9892	9653	467	213	3556	310	5081	125	41	96	222	55%
17	10x15	3195	13529	11951	3206	2167	5843	1860	10206	1939	52	1602	249	14%
18	10x15	2610	11759	12714	1334	685	6028	1008	7860	601	60	506	183	26%
19	10x15	2291	11939	14112	2884	1623	6993	1906	8913	941	43	912	74	44%
20	10x15	1992	12860	11416	1951	823	5507	1116	6437	681	19	576	80	30%

Table 7-3: Test results (assembly shop with single machine workstations) – objective value

# of	# of					modified				SB w/o re-opt		SB /w re-opt		
machines	oper.	SPT-1	SPT-2	LPT	EDD-O	EDD-O	EDD-J	FCFS	ATC(1,1)	obj.	time (sec)	obj.	time (sec)	improv.
5	5x10	662	1860	1334	191	74	536	154	1757	110	3	26	6	65%
5	5x10	220	1601	1963	367	25	244	96	1676	69	5	41	9	-64%
5	5x10	1012	3029	1313	493	287	317	234	2382	266	3	146	9	38%
5	5x10	338	2582	2513	372	23	857	81	1839	189	2	74	9	-222%
5	5x10	517	1483	2548	279	215	357	372	1588	40	3	39	7	82%
5	5x10	3450	8911	8305	2401	1925	3529	2356	9870	1204	16	1214	69	37%
5	5x10	2090	8185	10344	1467	1072	961	1307	6974	706	17	566	56	41%
5	5x10	1559	8804	10140	624	453	816	764	6961	543	13	333	76	26%
5	5x10	1188	8794	11237	918	619	1535	635	8689	575	22	372	62	40%
5	5x10	1777	9206	8787	1678	1149	1640	1522	8609	818	20	1080	72	6%
10	10x10	1121	6214	5120	603	761	2485	739	4718	341	9	281	35	53%
10	10x10	1598	8919	6186	1188	897	4451	1077	8730	1020	12	645	26	28%
10	10x10	628	4561	5065	699	646	2155	405	4008	198	13	224	28	45%
10	10x10	1741	6004	4991	1302	738	3334	901	6519	492	11	466	26	37%
10	10x10	601	4838	4141	765	501	2663	243	3573	268	11	283	38	-16%
10	10x15	915	6828	7524	836	529	3323	598	6033	621	36	581	77	-10%
10	10x15	412	4697	4663	969	465	2411	509	4350	279	46	155	130	62%
10	10x15	1003	7246	8149	840	516	2555	791	6870	395	48	231	193	55%
10	10x15	1560	9154	10261	1492	859	3907	1214	8218	774	82	535	72	38%
10	10x15	949	9159	7531	1233	367	3258	538	6052	442	42	200	126	46%

Table 7-4: Test results (two single m/c workstations; one 2-parallel m/c workstation; one batch machine)

Problem	# of operations	SPT	LPT	EDD-O	EDD-J	FCFS	ATC(1)	SB w/o re-opt		SB w/o re-opt		
								obj.	time (sec.)	obj.	time (sec.)	obj reduc.
1	5x10	174	207	198	165	174	85	8	1	8	3	91%
2	5x10	36	52	30	52	46	52	30	1	22	5	27%
3	5x10	391	191	125	96	71	173	20	5	20	15	72%
4	5x10	552	835	621	621	514	651	336	1	291	2	43%
5	5x10	45	168	45	60	160	80	5	1	5	10	89%
6*	5x10	601	497	601	601	601	393	131	5	123	18	69%
7*	5x10	758	800	824	784	648	588	270	1	228	3	61%
8*	5x10	624	695	277	397	362	596	109	0	105	16	62%
9*	5x10	454	953	870	514	369	856	70	5	18	6	95%
10*	5x10	359	387	499	349	459	297	165	6	145	16	51%
11	10x10	377	443	471	314	467	437	138	61	125	80	60%
12	10x10	1016	1264	1043	1204	1247	1197	417	84	405	99	60%
13	10x10	803	1094	1051	981	846	2050	278	68	299	235	63%
14	10x10	265	273	131	313	505	367	129	82	62	240	53%
15	10x10	779	341	585	940	413	344	46	3	46	16	87%
16*	10x10	2084	2280	1728	1659	2091	2473	1321	87	1007	110	39%
17*	10x10	3575	4159	4126	3067	3254	3569	2761	101	1814	252	41%
18*	10x10	3806	4690	4100	3291	3971	3665	1680	87	1583	102	52%
19*	10x10	3475	4273	2990	2842	3424	2651	1530	96	921	281	65%
20*	10x10	2724	3072	2721	2576	2836	2917	1084	5	910	115	65%

Table 7-5: Test results (four single m/c workstations; two 2-parallel m/c workstations; one batch machine)

Problem	# of operations	SPT	LPT	EDD	EDD-J	FCFS	ATC(1)	SB w/o re-opt		SB w/o re-opt		
								obj.	time (sec.)	obj.	time (sec.)	obj reduc.
1	10 x 10	361	270	273	162	233	315	134	26	57	151	65%
2	10 x 10	482	633	431	425	506	1002	60	14	35	141	92%
3	10 x 10	1050	945	1156	930	692	1198	408	20	224	181	68%
4	10 x 10	647	714	631	441	495	887	625	29	270	130	39%
5	10 x 10	146	133	599	86	123	637	31	23	25	127	71%
6*	10 x 10	1554	1746	1340	1762	1207	3036	701	39	699	122	42%
7*	10 x 10	1477	1872	2116	1393	1171	2277	598	27	596	182	49%
8*	10 x 10	1520	1550	1922	1081	1196	2120	927	26	748	191	31%
9*	10 x 10	1678	1691	2264	1762	1833	2944	1764	25	1302	202	22%
10*	10 x 10	1539	2205	3117	1247	1584	2581	519	24	483	200	61%
11	10 x 15	903	742	1004	678	575	939	302	136	231	672	60%
12	10 x 15	379	445	396	328	416	402	58	92	72	716	78%
13	10 x 15	742	481	272	254	224	612	245	94	84	462	63%
14	10 x 15	410	432	464	367	567	572	195	20	205	577	44%
15	10 x 15	1269	1175	1716	1107	1613	1151	1176	194	726	687	34%
16*	10 x 15	2436	3022	2810	2187	2765	2454	2027	111	1624	894	26%
17*	10 x 15	1112	1394	1487	1015	1149	2063	621	127	541	698	47%
18*	10 x 15	1267	1797	1346	1379	1610	1426	1307	23	1041	651	18%
19*	10 x 15	1536	1805	1461	1361	1322	1197	1033	21	821	744	31%
20*	10 x 15	1898	2293	2242	1521	2179	1506	1341	25	1312	939	13%

Table 7-6: Test results (multiple types of workstations assembly shop with sequence dependent setup)

# of	# of							SB w/o re-opt		SB /w re-opt		
machines	operations	SPT	LPT	EDD	EDD-J	FCFS	ATC(1,1)	w/o re-opt	time (sec.)	/w re-opt	time (sec.)	reduc.
5	5 x 10	274	341	301	274	283	292	135	5	107	8	61%
5	5 x 10	209	109	129	205	205	157	52	5	52	15	52%
5	5 x 10	391	284	207	134	67	165	20	2	20	8	70%
5	5 x 10	693	785	864	689	774	778	330	6	369	8	46%
5	5 x 10	213	426	264	220	259	153	53	6	53	9	65%
5	10 x 10	1102	745	1041	707	710	986	433	69	314	86	56%
5	10 x 10	2669	2190	3221	1811	1860	1954	961	51	863	306	52%
5	10 x 10	1658	1970	2928	1656	1547	1689	993	52	798	269	48%
5	10 x 10	886	1314	1377	664	686	1841	333	84	325	104	51%
5	10 x 10	841	1001	941	1358	794	1133	1142	64	427	204	46%
9	10 x 10	485	485	1031	410	335	1328	136	36	136	133	59%
9	10 x 10	481	951	782	503	460	682	242	26	234	151	49%
9	10 x 10	1157	1251	1304	1047	1091	1473	410	32	343	214	67%
9	10 x 10	826	826	879	828	718	1234	560	11	568	69	21%
9	10 x 10	232	352	747	216	210	1039	129	20	63	144	70%
9	10 x 15	2345	1874	1147	1382	1039	1930	805	141	715	625	31%
9	10 x 15	1700	1245	984	1035	1786	701	691	145	556	539	21%
9	10 x 15	879	1087	630	712	828	763	472	125	362	501	43%
9	10 x 15	1097	1075	891	771	888	1164	1137	53	752	650	2%
9	10 x 15	1733	1544	2522	1569	1740	1911	1725	165	1048	838	32%

CHAPTER 8

ASSEMBLY TYPE JOB-SHOPS WITH OTHER OBJECTIVES

Until this moment, the only objective that we have emphasized is minimizing the weighted tardiness. In this chapter, we will discuss the development of shifting bottleneck heuristics for other objectives. The objectives can be categorized into two types. The first type is the objective that is related to the job completion time. There is a small modification on the heuristic for this type. The other is the objective that is not directly related to the job completion time such as the number of late jobs.

8.1 Job Completion Time Related Objectives

This type of objectives includes one related to makespan, flow-time, lateness, tardiness, and earliness related objective. This type of objectives can be modeled with the modified disjunctive graph. The objective value can be determined directly from the graph. Therefore, the SB concept can be applied. In the following, we discuss the methods for sequencing the workstation when other types of objectives are concerned.

8.1.1 CPI on Other Objectives

Most of the heuristics developed in the previous chapters are based on critical path improvement technique. This technique determines the operations that contribute to the completion time of the jobs. Then the sequence improvement is done by moving these operations in a way such that it will reduce the jobs completion time. Therefore, this technique can be applied to other objectives such as minimize the makespan (maximum job completion time), minimize the weighted completion time, minimizing job lateness, minimizing weighted lateness, etc. By modifying the objective function, CPI should be able to adapt to the new environment. The details of CPI for some objectives are discussed below.

Minimizing the Makespan

This objective is the most common objective. It considers the completion time of all the jobs processed. Minimizing the makespan can be viewed as maximizing the machine utilization as we try to pack the processing close together.

To apply CPI on this objective, only the critical path related to the job that contributes the highest completion time will be focused. The completion time of this job determines the objective value. The rest will be discarded. However, the branch and bound method developed by Carlier is more suitable for this objective.

Minimizing the Maximum Job Lateness

Similar to minimizing makespan objective, single critical path is focused. The job lateness is determined from the difference between the job completion and the job due date. After the critical operations have been determined, we check for the moves that will provide benefit. The objective function used in this step is $\max_{j}(C^{(j)} - d^{(j)})$ where $j = 1,\dots,n$.

Minimizing the Weighted Completion Time or Minimizing the Weighted Lateness

Proposition 1: The minimizing weighted completion time and minimizing the weighted lateness objectives are equivalent.

Proof:

$$\min \sum_{j=1}^{m} w^{(j)} L^{(j)} = \min \sum_{j=1}^{m} w^{(j)} (C^{(j)} - d^{(j)})$$

$$= \min \left(\sum_{j=1}^{m} w^{(j)} C^{(j)} - \sum_{j=1}^{m} w^{(j)} d^{(j)} \right)$$

$$= \min \sum_{j=1}^{m} w^{(j)} C^{(j)} - K, \qquad \text{where } K \text{ is a constant}$$

$$= \min \sum_{j=1}^{m} w^{(j)} C^{(j)}$$

For these two objectives, we may apply CPI method by modifying the objective function on determining benefits of moving operations.

Minimizing the Maximum Weighted Lateness

This objective is focusing on high priority jobs. These jobs are assigned with high penalty costs. When a high weight job is delayed, it contributes high objective value. Similar to other objectives, CPI method may be applied after modifying the objective function. However, only the critical path that passing through the most tardy job need to be focused.

Limitation of CPI Method

CPI method is based on moving operations to reduce the job completion time. Therefore, CPI cannot be applied with the objective that does not improve when the job completion time reduces, *ie*. minimizing the job earliness, minimizing the number of late jobs, etc.

8.1.2 Combination of Shifting Bottleneck and Simulated Annealing

In order to accommodate general objectives, we need a heuristic that is highly flexible. Neighborhood search technique can provide this type of flexibility. However, the computation time of the neighborhood search seems to be rather high when considering a large size problem. Therefore, we may apply the combination of SB and neighborhood search technique on solving the problem. SB provides the means to decompose the large size problem into small size ones. These small problems could be solved by SA in an acceptable amount of time.

8.1.2.1 Simulated Annealing (SA): SA is a type of neighborhood search technique that can be applied to various types of combinatorial optimization problems. It was proved that this method can generate a very good solution comparable to a specially designed heuristic. This does not consider the computation time.

8.1.2.2 Neighborhood Structure: The efficiency of the SA technique depends on the designed of the neighborhood structure and parameters determination. In the following, we discuss the neighborhood structure. We propose neighborhood structures for the three types of workstation. However, we skip their performance evaluation.

Single Machine Workstation

The neighborhood can be generated from the current sequence by interchanging two operations on the sequence. There are two issues that should be considered. First, operations dependency has to be checked to ensure that the generated sequence will be feasible. Second, the critical path should pass through one of the interchanging operations. Swapping two operations that are not on the critical path will never improve the objective value.

Parallel Machine Workstation

The neighbor structure for parallel machine workstation can be based on string sequence. First, transform the parallel machine sequences to a string sequence. Then apply the interchanging method similar to the single machine case. Check for the precedent constraints violation. After a new sequence is generated, transform it back to parallel machine sequences.

Batch Machine Workstation

Similar to parallel workstation, the neighbor of the batch machine sequence can be developed by converting the batch sequence to a string sequence. Apply the interchange method. Then, use the batching heuristic presented in chapter 6 to assign the operations to the batches.

8.1.2.3 The Heuristic Development Guideline: The following procedure is a guideline to develop the SA for sequencing the workstations. The steps in SB remain the same except the bottleneck sequencing phase. SA method will replace the workstation sequence heuristics.

First, generate a feasible sequence from a priority rule such as EDD-J, EDD-O, SPT, ATC, etc. Then, generate *n* number of neighbors (*n*=30). Use the statistical analysis to set the temperature parameter. After this initialize stage, the annealing stage can be started. The neighbor will be generated using the provided structure depending on the type of the workstation. If the new neighbor provide a better sequence, accept it as a new generating point. If it does not, accept it with a probability *p*. Continue until no improvement is found for the last *m* repetitions. Report the best sequence found.

8.2 Minimizing Weighted Number of Tardy Jobs

In job shop scheduling, minimizing number of tardy jobs is one of the important objectives beside minimizing makespan and tardiness. This problem was well studied on one machine problem. A well known heuristic by Moore (1968) is proved to find the optimal solution for $1||\Sigma U_j$. The $1|| \Sigma w_j.U_j$ is proved to be an NP-hard problem by Karp (1972). Approximation algorithms were developed by Sahni (1976), Gens and Levner (1981), Ibarra and Kim (1978). In the following, we proposed an extended shifting bottleneck heuristic on weighted tardiness in chapter 7 to accommodate this objective.

The classical job shop problem is described as follows. There are *n* jobs to be processed on *m* machines. Each job has a pre-defined release time and due date. The machine can process one job at a time. The processing on the machine is called an *operation*. The machine routing (series of operations) is fixed and known in advance. The objective of the problem is to find the sequence of the operation of each machine that will minimize the number of tardy jobs. The mathematical formulation for this problem is as follows.

$$\text{Min.} \quad \sum_{j=1}^{n} w_j U_j$$

subject to

$M * U_j \geq T_j,$	$j = 1, \ldots, n,$
$T_j \geq t_j^* - d_j,$	$j = 1, \ldots, n,$
$T_j \geq 0,$	$j = 1, \ldots, n,$
$t_t - t_s \geq p_s,$	$(s,t) \in A,$
$t_s \geq r_s,$	for all $s,$
$t_t - t_s \geq p_s$ or $t_s - t_t \geq p_t,$	$s, t \in E_k, k = 1,.., m,$
$U_j = 0$ or $1,$	$j = 1 .. n,$

where,

M	is a large number,
U_j	equals 1 if job j is late or 0 if job j is on time,
T_j	is the tardiness of job $j,$
t_j^*	is the completion time of job j (starting time of the sink node),
d_j	is the due date of job $j,$
t_s	is the starting time of operation $s,$
p_j	is the processing time of operation $j,$
A	is the set of pairs of operations that have precedent relationship (conjunctive arcs set),
r_j	is the releasing time of job $j,$
E_k	is the set of operations that need to be processed on machine k (disjunctive arcs set).

8.2.1 The Concept

The method in solving job shop scheduling with minimize number of tardy jobs is based on the following two propositions.

Proposition 1: If no late job schedule exists, the algorithm to minimize the weighted tardiness or maximum tardiness will provide the optimal schedule for minimizing the number of tardy jobs.

Proof: If no late job schedule exists, the algorithm to minimize the weighted tardiness or maximum tardiness will provide a schedule with objective value of zero. No job will be late. Hence, they provide the optimal schedule for minimizing the number of tardy jobs.

Proposition 2: Only re-scheduling the jobs that have operations on the critical path(s) can improve the objective value (weighted number of tardy jobs).

Proof: Re-schedule the operations that are not on the critical path(s) will not change the objective value unless they are on the critical path of the new schedule. If they are, the objective value will increase.

From the above propositions, the heuristic based on shifting bottleneck heuristic can be developed. The structure of SB is quite similar to one in chapter 7. The major modification is on workstation sequencing.

8.2.1.1 Bottleneck Determination: We can determine the bottleneck workstation from the following steps. First, select a focused workstation. Fix the sequences of the other workstations. Then, find the sequence that provide the lowest objective value. Save this value as the bottleneck index. Repeat the steps but change the focused workstation until indices are determined for all workstations. The bottleneck workstation is the one that has the highest index.

To improve the computation time, rough estimation will be applied to this step. TER, PEDD-O or BEDD-O+Batching are used instead of lengthy calculation ones. Though, fast heuristics may not correctly determine the bottleneck workstation every time, they determine the bottleneck or near bottleneck workstation that is sufficient for the shifting bottleneck heuristic.

8.2.1.2 Removing Job: When we solve the workstation sequencing problem and found that there is no possible alternative to sequence the operations in such a way that no job is delayed. We know that at least one job will be delayed. We can select to delay any job. The one to select is the one that has high processing time on the critical path related to the late jobs. Delaying this job will create large space on highly utilized workstations for the other jobs. As delaying a job will increase the objective value equally no matter how late it is, it is better to put the job to the last position in the sequence on all machines. In other words, we can discard this job and its operations from the problem. After the sequences have been fixed for the remaining jobs, this job will be re-considered. The operations belong to it will be processed last on the sequences.

8.2.1.3 Workstation Sequencing: After we have selected the bottleneck workstation, the sequence for that workstation will be fixed by applying the heuristic on chapters 4, 5 or 6 to the decomposed problem according to the structure. If the sequence generated does not provide the job lateness, there is no late job. We do not need an extra work. Otherwise, find the job that has the highest total processing time on the critical paths that related to late jobs. Remove that job from the problem. After all the operations belong to the job have been removed, re-apply the heuristic. Repeat the steps until there is no delay job.

If SB on min $\Sigma w_j T_j$ provides a solution with no late job, this schedule is optimal. If late jobs exist, some jobs among the late jobs need to be postponed. As a result, the rest of the late jobs have potential to finish their processing before their due dates. Hence, it is reasonable to postpone the job that will provide the greatest slacks for the rest. The late job that has the longest weighted processing time on the critical path(s) is chosen. This job is removed from our consideration. Later, it will be scheduled as the last job with the lowest priority. The new job completion times can be determined. If there is no late job, this new schedule is optimal.

8.2.2 The Heuristic

In proposition 1, we suggest that both heuristics that minimize the maximum tardiness or the weighted tardiness provide the optimal schedule if no late job schedule exists. The reason that we choose the heuristic that minimize the weighted tardiness is as follows. The heuristic that minimize the maximum tardiness tends to balance the lateness among all the jobs. It has potential to create more critical paths with less processing time on them for each job. Hence, removing the late job according to step 2 will provide less slacks for the rest.

1. Decompose the problem into sub-problems. Each sub-problem is a workstation sequencing problem with tails.
2. Determine the bottleneck workstation
3. Sequence the bottleneck workstation. If tardy job exists, remove the job that has the longest processing time on the critical paths from the problem.
4. Apply local re-optimization steps.
5. Apply global re-optimization steps.
6. Append the operations of the removed jobs from step 3 to the end of the sequences.

Example 8-1: We use the same problem illustrated in chapter 7 (Example 7-1). However, the objective changes from minimize weighted tardiness to minimize number of late jobs.

Iteration 1

Job: 1, 2, 3

Unscheduled machines: 1, 2, 3

Scheduled machines: ∅

Bottleneck machine: 1

The sequence after CPI was applied is {M1: 6, 8, 2, 1, 11, 13} which has objective value of 20.

Remove: Job 2

As the weighted tardiness is greater than 0, there is at least one late job. Job 1 is late and the summation of the length on critical paths for job 1 is 4+3+6+6 = 19. Job 2 is late with the total lengths on critical path of 5+3+4+2=12. Therefore, job 1 will be removed.

Iteration 2

Job: 1, 3

Unscheduled machines: 1, 2, 3

Scheduled machines: ∅

Bottleneck machine: 1

It is determined by TER. We applied CPI to the sequence. The result is {M1: 2, 1, 11, 13} which has objective value of 0.

Iteration 3

Job: 1, 3

Unscheduled machines: 2, 3

Scheduled machines: 1

Bottleneck machine: 3

After applied CPI, the result is {M3: 3, 5, 14} with objective value of 2.

Remove: Job 3

As the weighted tardiness is greater than 0, there is at least one late job. From the determined sequence, job 3 is late and the summation of the length on critical path related to job 3 for job 1 is 4+3+6 = 13. The one for job 3 is 6+4+5+3=18. Therefore, job 3 will be removed.

Iteration 4

There is only job 1 left. Therefore, the number of late job = 1. Then, we add jobs 2 and 3 to form the final sequence.

Sequence summary: {M1: 2, 1, 6, 8, 11, 13; M2: 4, 7, 10, 12, 15; M3: 3, 5, 9, 14}

8.3 Numerical Examples

The same set of problems used in testing the heuristic in chapter 7 is reused for testing modified SB for other objectives. We test the performance of three versions of SB on makespan, weighted flow time, and weighted lateness objectives. They show improvement on most cases.

On minimizing maximum completion time, FCFS rule performs very well when shops have light loads. The performance of SB improves as the problem becomes more congested. As the number of jobs increases, the weighted flow time increases. The improvement of the heuristic is determined by dividing the difference between the best objective value from other priority rules and the objective value from SB by the objective value from SB. Therefore, the improvement on the objective seems to be reduced as the number of jobs grows. Table 8-1 shows the improvement of the objective value when applying SB. The test results are shown in Table 8-2 to Table 8-7.

Table 8-1: Average improvement of SB heuristics on 3 different objectives

	Makespan (Cmax)	Weighted flow time	Weighted lateness
5 machines	3%	9%	40%
10 machines	5%	3%	46%

Table 8-2: Minimize maximum completion time of jobs (5 machines)

	# of					modified				SB		
Problem	oper.	SPT-1	SPT-2	LPT	EDD-O	EDD-O	EDD-J	FCFS	ATC(1)	obj.	time (sec)	improv.
1	5x10	142	1596	1183	143	135	181	131	259	133	2	-2%
2	5x10	132	195	217	151	119	148	128	214	124	3	-4%
3	5x10	191	301	235	185	167	181	158	288	160	3	-1%
4	5x10	158	271	258	162	142	213	141	271	140	4	1%
5	5x10	168	232	302	177	148	185	140	208	140	3	0%
6	5x10	236	345	423	245	227	311	217	379	220	9	-1%
7	5x10	231	372	439	234	204	287	206	408	191	7	6%
8	5x10	259	397	444	233	222	336	210	432	210	9	0%
9	5x10	253	439	461	248	238	339	227	467	208	11	8%
10	5x10	292	395	397	258	240	358	240	435	220	9	8%
11	10x10	254	457	442	255	236	341	231	410	225	28	3%
12	10x10	294	466	482	286	260	308	262	453	256	45	2%
13	10x10	294	515	543	259	249	327	259	493	236	38	5%
14	10x10	245	469	525	270	238	308	238	467	230	52	3%
15	10x10	282	514	461	280	258	337	258	479	252	39	2%
16	10x10	351	555	586	381	325	496	322	596	300	81	7%
17	10x10	356	606	720	367	328	546	344	572	296	53	10%
18	10x10	386	689	727	344	319	514	325	698	299	70	6%
19	10x10	398	624	636	343	333	484	357	566	317	138	5%
20	10x10	368	652	719	382	335	503	339	681	297	53	11%

Table 8-3: Minimize maximum completion time of jobs (10 machines)

Problem	# of oper.	SPT-1	SPT-2	LPT	EDD-O	modified EDD-O	EDD-J	FCFS	ATC(1)	SB obj.	SB time (sec)	SB improv.
1	10x10	151	330	280	154	148	284	151	318	137	16	7%
2	10x10	164	352	259	173	139	311	135	281	131	22	3%
3	10x10	152	282	297	152	149	246	143	262	135	19	6%
4	10x10	161	322	285	187	146	304	145	313	132	16	9%
5	10x10	176	339	315	173	159	290	140	327	128	17	9%
6	10x10	249	406	457	263	247	403	223	379	225	24	-1%
7	10x10	255	436	395	284	227	454	238	429	216	31	5%
8	10x10	240	358	453	236	216	321	224	415	211	33	2%
9	10x10	222	427	458	271	213	456	218	429	205	41	4%
10	10x10	228	333	482	254	223	406	216	377	213	54	1%
11	10x15	246	519	497	267	249	427	254	551	233	33	5%
12	10x15	230	463	461	258	224	426	223	521	213	43	4%
13	10x15	223	461	456	255	218	374	217	491	218	44	0%
14	10x15	228	415	414	252	219	388	224	387	205	41	6%
15	10x15	229	475	446	265	232	438	230	466	220	72	4%
16	10x15	309	670	657	327	308	680	294	631	274	173	7%
17	10x15	316	659	616	402	331	650	278	699	262	182	6%
18	10x15	313	712	791	348	267	691	270	773	248	128	7%
19	10x15	321	656	608	347	301	676	284	621	264	72	7%
20	10x15	329	740	659	395	316	646	305	611	302	84	1%

Table 8-4: Minimize weighted flow time (5 machines)

	# of					modified				SB		
Problem	oper.	SPT-1	SPT-2	LPT	EDD-O	EDD-O	EDD-J	FCFS	ATC(1)	obj	time (sec)	improv.
1	5x10	1194	2590	2177	1006	907	1331	978	2346	865	3	5%
2	5x10	961	2228	2584	1045	760	961	801	2242	674	5	11%
3	5x10	2001	3881	2323	1567	1373	1375	1265	3062	1153	5	9%
4	5x10	1258	3485	3339	1195	869	1718	980	2911	884	4	-2%
5	5x10	1263	2185	3215	934	892	1002	1028	1410	752	4	16%
6	5x10	2351	3390	4437	1776	2525	3000	2768	4137	1391	9	22%
7	5x10	2067	4500	6387	1911	1717	2256	1849	4118	1446	6	16%
8	5x10	2519	4105	5074	2268	2186	2967	2197	3561	1774	6	19%
9	5x10	2768	5445	5142	2274	2056	2539	2267	3864	1830	11	11%
10	5x10	2131	3187	3916	2114	1889	2703	1977	2715	1503	10	20%
11	10x10	4243	9743	9490	3642	3239	4818	3585	6773	2626	33	19%
12	10x10	3152	8973	11061	2814	2326	2480	2572	6996	2204	41	5%
13	10x10	2919	9758	10887	1879	1681	2138	1878	7144	1552	47	8%
14	10x10	2224	9426	11942	2211	1838	2650	1973	7116	1805	33	2%
15	10x10	2873	10100	9830	2985	2454	3083	2659	8235	2512	37	-2%
16	10x10	5253	11368	11255	4608	3799	5776	4035	9397	3648	61	4%
17	10x10	6436	13562	15360	4983	4132	7327	4544	9206	4107	64	1%
18	10x10	6841	15188	15391	4744	4172	6859	4804	10455	3988	81	4%
19	10x10	6183	11771	12886	3723	3460	5206	4118	7038	2949	100	15%
20	10x10	5388	10126	12497	4742	4143	6032	4162	8765	3983	52	4%

Table 8-5: Minimize weighted flow time (10 machines)

	# of					modified				SB		
Problem	oper.	SPT-1	SPT-2	LPT	EDD-O	EDD-O	EDD-J	FCFS	ATC(1)	obj.	time (sec)	improv.
1	10x10	2565	7352	6542	2112	1961	3848	2124	5419	1801	19	8%
2	10x10	3609	10318	7745	3163	2643	6330	2777	6373	2537	18	4%
3	10x10	1909	5561	5932	1790	1716	3407	1655	3706	1422	21	14%
4	10x10	3202	7608	6496	2995	2225	4841	2158	5769	2107	22	2%
5	10x10	2019	5802	5601	2061	1852	3863	1635	3563	1600	23	2%
6	10x10	3476	6970	9945	3608	3134	5103	2859	5058	2836	16	1%
7	10x10	4785	9441	8719	4764	3457	5363	3266	6106	3217	22	2%
8	10x10	5071	9617	11239	4318	4076	5137	4139	7221	3542	38	13%
9	10x10	4769	8918	11397	5684	4279	7746	4290	7138	3771	50	12%
10	10x10	5050	8701	13592	5570	4627	7824	4868	8187	4950	60	-7%
11	10x15	2633	8423	8973	2507	2436	4991	2365	7084	2377	28	-1%
12	10x15	1762	5724	5689	2041	1541	3577	1570	5001	1425	34	8%
13	10x15	2487	8482	9060	2348	2051	4123	2241	6410	2042	59	0%
14	10x15	3717	10758	11740	3451	2909	5629	3201	7573	3051	50	-5%
15	10x15	3087	10789	9233	2985	2492	4954	2591	6964	2162	74	13%
16	10x15	3376	12216	11977	2720	2400	5820	2519	7405	2625	81	-9%
17	10x15	6424	16758	15180	6435	5396	9033	5089	13435	5083	205	0%
18	10x15	5445	14594	15549	4133	3508	8848	3810	10635	3509	144	0%
19	10x15	5681	15349	17522	6266	5021	10351	5264	12323	5023	57	0%
20	10x15	5162	16030	14586	5121	3979	8665	4286	9607	3484	77	12%

Table 8-6: Minimize the maximum weighted lateness (5 machines)

Problem	# of oper.	SPT-1	SPT-2	LPT	EDD-O	modified EDD-O	EDD-J	FCFS	ATC(1)	SB obj.	SB time (sec)	SB improv.
1	5x10	102	500	321	20	0	172	63	417	0	3	0%
2	5x10	80	415	595	84	0	75	40	475	9	2	-
3	5x10	332	808	388	140	60	120	56	664	20	5	64%
4	5x10	65	870	676	85	0	340	40	620	0	5	0%
5	5x10	250	550	980	69	70	138	164	188	51	4	26%
6	5x10	357	654	792	586	135	387	171	567	120	10	11%
7	5x10	280	912	1408	260	140	472	180	760	38	7	73%
8	5x10	345	955	920	275	220	790	264	468	152	11	31%
9	5x10	414	1172	1240	295	245	555	280	693	100	11	59%
10	5x10	264	470	1040	380	290	880	290	412	126	12	52%
11	10x10	450	1228	1695	455	360	885	348	819	150	43	57%
12	10x10	415	1360	1475	250	170	312	270	892	92	35	46%
13	10x10	360	1570	1930	120	90	324	144	924	50	41	44%
14	10x10	156	1100	1768	152	76	228	140	837	76	30	0%
15	10x10	255	1800	1810	355	245	525	245	1020	171	44	30%
16	10x10	681	1760	1374	568	344	896	456	1071	250	59	27%
17	10x10	832	2044	2340	460	430	1140	635	1230	244	58	43%
18	10x10	1120	2115	2665	432	392	968	590	890	216	94	45%
19	10x10	627	1479	1710	895	650	1130	520	1088	151	109	71%
20	10x10	625	2020	2290	484	328	1200	392	924	196	54	40%

Table 8-7: Minimize the maximum weighted lateness (10 machines)

Problem	# of oper.	SPT-1	SPT-2	LPT	EDD-O	modified EDD-O	EDD-J	FCFS	ATC(1)	SB		
										obj.	time (sec)	improv.
1	10x10	260	1330	1065	145	64	735	105	585	58	18	9%
2	10x10	305	1245	728	215	140	905	140	633	96	16	31%
3	10x10	152	820	724	80	80	368	60	424	25	18	58%
4	10x10	265	1070	780	196	105	656	99	636	60	15	39%
5	10x10	210	1025	830	195	125	780	74	387	34	18	54%
6	10x10	312	915	1448	300	282	702	210	432	101	28	52%
7	10x10	530	1196	1275	535	315	1010	225	500	95	24	58%
8	10x10	550	1110	1780	284	285	616	450	609	145	22	49%
9	10x10	385	1265	1485	555	340	988	380	840	129	51	62%
10	10x10	372	792	1775	415	325	1025	385	736	228	48	30%
11	10x15	215	1130	1325	140	140	748	170	856	68	26	51%
12	10x15	152	756	1005	160	81	725	99	561	33	37	59%
13	10x15	180	1148	1216	155	80	750	105	674	39	36	51%
14	10x15	295	1695	1500	250	190	920	230	995	117	43	38%
15	10x15	228	1785	1820	200	124	930	200	710	35	51	72%
16	10x15	228	2264	2024	120	82	826	132	696	36	142	56%
17	10x15	696	1956	1760	608	392	1161	400	1425	232	158	41%
18	10x15	708	2304	2348	312	183	1716	220	1293	132	158	28%
19	10x15	560	2170	2036	725	495	2345	324	1232	212	72	35%
20	10x15	690	2745	2032	357	180	1380	320	952	93	94	48%

CHAPTER 9

SCHEDULING INFORMATION SYSTEM

In this chapter, we describe the scheduling system for assembly shop environment. First, we discuss the general picture of the information system for production facilities. Then, we discuss LEKIN which is an assembly job-shop scheduling system developed for research purposes. The system is focused on the graphic user interface as well as scheduling tools development. We employ object oriented design and programming on the developing phase.

9.1 Information System for Production Facilities

In the competitive world, the quality and the timing of the information can justify success or failure of a company. The company success relies on the right decision at the right time. The information system plays a major role as a decision support tool for the management. In the following, we analyze the production information system based on the company functions. We may divide a company into four major departments as follows.

(i) Marketing department

(ii) Production department

(iii) Distribution department

(iv) Control and support department

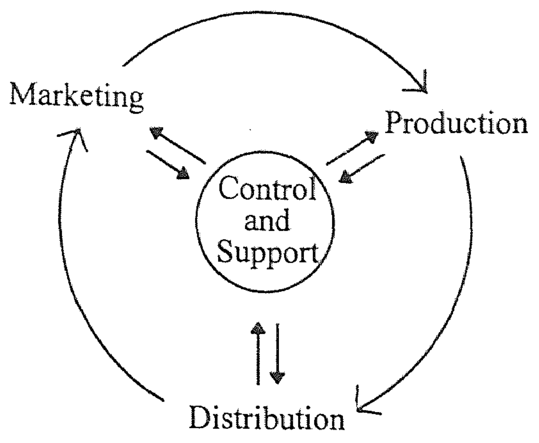


Figure 9-1: Department relationship

Figure 9-1 shows the relationship among the departments. Each department will respond to specified functions. A list of main functions is shown below.

(i) Sales management

(ii) Product design

(iii) Process design

(iv) Production scheduling

(v) Manufacturing control

(vi) Inventory

(vii) Transportation & distribution

(viii) Resource planning

(ix) Supporting system

The functions' relationship is described in Figure 9-2. Information transfers are shown in arrows and dotted lines.

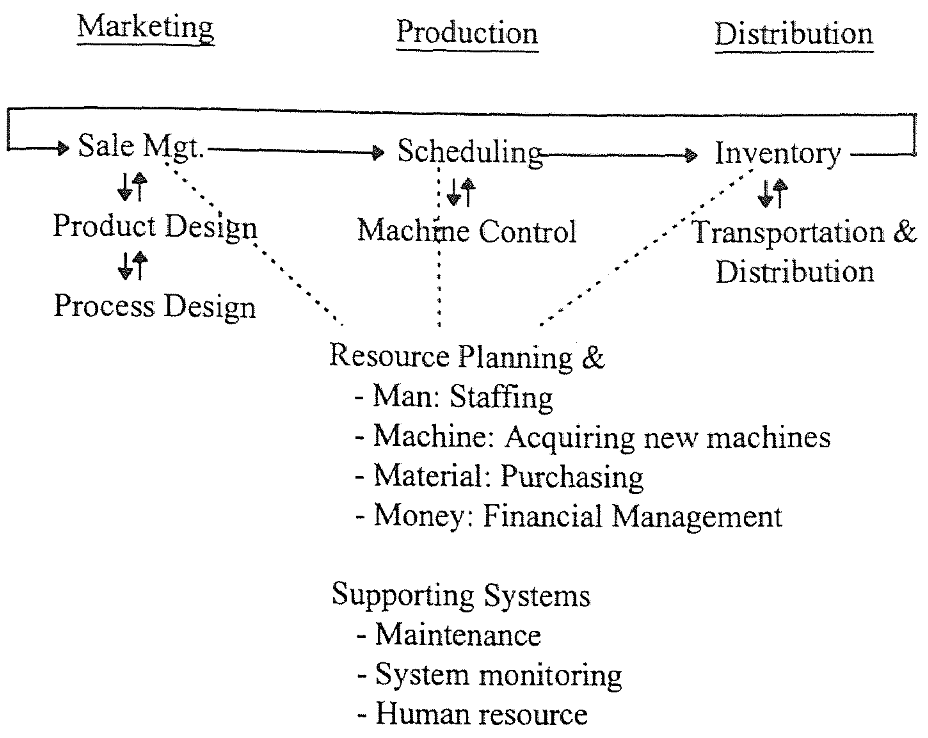


Figure 9-2: Relationship of functions

Information System

The information transfers between Marketing, Production and Distribution are one-direction transfers. The bi-directions transfers are used between Control and Support department with others. The feed backs from departments allow the management to adjust the plans when unexpected events occur.

Marketing

In the system, marketing functions cover product design, process design and sales management. This department can be viewed as an interfacing between the company and the customers. Due to this fact, marketing personnel has the best knowledge on customer needs among all departments. Therefore, product design Therefore, they should be the people that design products to satisfy the customers' needs. CAD and analysis packages are necessary software for product designing.

On the design stage, the product designer needs to concern about transforming his design to the production line. Expert system such as knowledge based could be applied here. The knowledge based system can recommend an appropriate process for the production by using its database stored with knowledge from experts. This information is, then, sent to the production department.

Sales management information provides the product manager with current market status, historical data, forecasted information and "what if?" simulation. The system should apply the state of the art computer learning capability with a powerful database to create a decision simulation tool. This tool could guide product managers in making decision in the stochastic environment. If further information is needed, he can send a request to the supporting system. The supporting system will provide data query.

Production

Information from Marketing will be sent to production department. The important information is order information and product routing. The resource availability is provided from the control & support department. The schedule will be made. The information will be sent to machine control system. This system controls the machine operating sequences. Further details on production scheduling will be discussed in the next section.

Distribution

After products are fabricated, they are ready to be sent to warehouse or customers. Inventory system will trace and control the level of stocks to minimize cost and maximize customer satisfactions. This system is linked to transportation and distribution system seamlessly. The information on customers and orders will be transferred to transportation and distribution system when the products need deliveries. The transportation and distribution system will manage the material handling equipment, trucks, routing and documents.

Control and Support

Control and Support department is the center of information system. It should be able to provide job status for customer service staffs or market trend for inventory manager. The appropriate plan to control the resources should be developed in real-time. Four important resources are man, machine, material and money. The system will request the information from every department. Then, the decision will be justified based on available resources.

Beside direct functions, support functions are also important. These functions should be served to the whole company. Supporting tasks are as follows.

- Maintenance: machines, computer system, material handling system, trucks, etc.
- System monitoring: sales, costs, machine status, job status, package status, truck status, etc.
- Human resources: health care, benefits, recruitment, etc.

Scheduling System

The following list includes the key issues in developing a DSS in modeling scheduling systems. The structure of the system is explained in Figure 3.

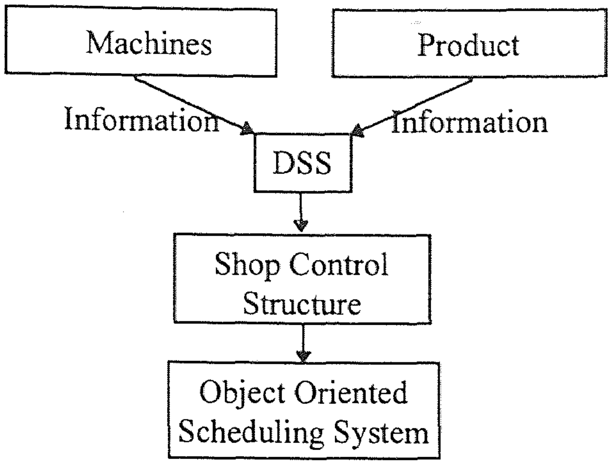


Figure 9-3: Structure of the system and its linkage

9.2 Object Oriented Programming

Object oriented programming (OOP) is a new paradigm for computer programming. Instead of trying to mold the problem into something familiar to the computer, it adapts the computer to the problem. OOP supports three key features:

Abstract Data Typing: The programmer can create new data types to adapt to his needs. The structure together with its operations (functions) are incorporated into the new data type which is defined as *object*.

Inheritance: Once an object is defined, it can be the basis for a new data type.

Polymorphism: The object can be designed to respond to the appropriate message (command). For example, when *draw* is called, *circle* object will draw a circle while *triangle* object will draw a triangle.

By using OOP, the software development is more organized. The modification and maintenance of the software is simplified. OOP promotes the codes reusable and enhance the encapsulating ability. Thus, it is easier to work on the complex system than ever.

9.3 Objects and Classes

Machine Shop System is the facility and management that produce the product according to the given demand. It can be divided into two sections -- hardware and software. On the hardware side, there are *machines*. The machines in the factory shop that perform similar tasks are normally grouped into *workstations*. Each workstation is assigned to perform a specified type of work.

An *order* is the demand from a customer. The customer specifies the product specifications, delivery time, quantities, and price in his order. The order is broken down into *jobs* on the production level. Each job is a product or a batch of products that need to be processed on various machines according to its routing. The processing on a machine is called *operation*. It is the scheduler task to manage the machines to produce the products according to the given demands. He needs to issue the production *sequences* for each machine and send them to the operators.

On the above paragraphs, the italic words are basic objects needed for developing the scheduling information system. Figure 9-4 shows their relations. There are many more classes and objects used in the system which will be discussed later.

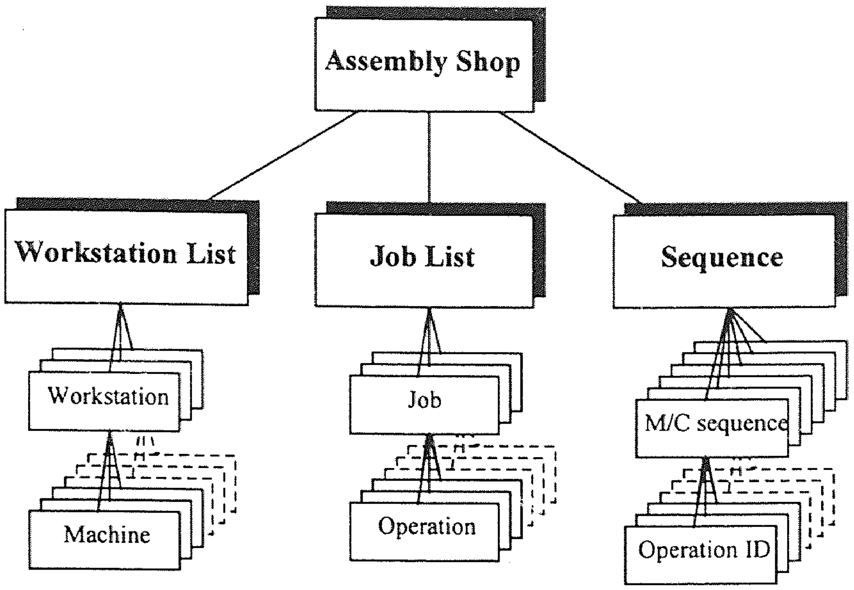
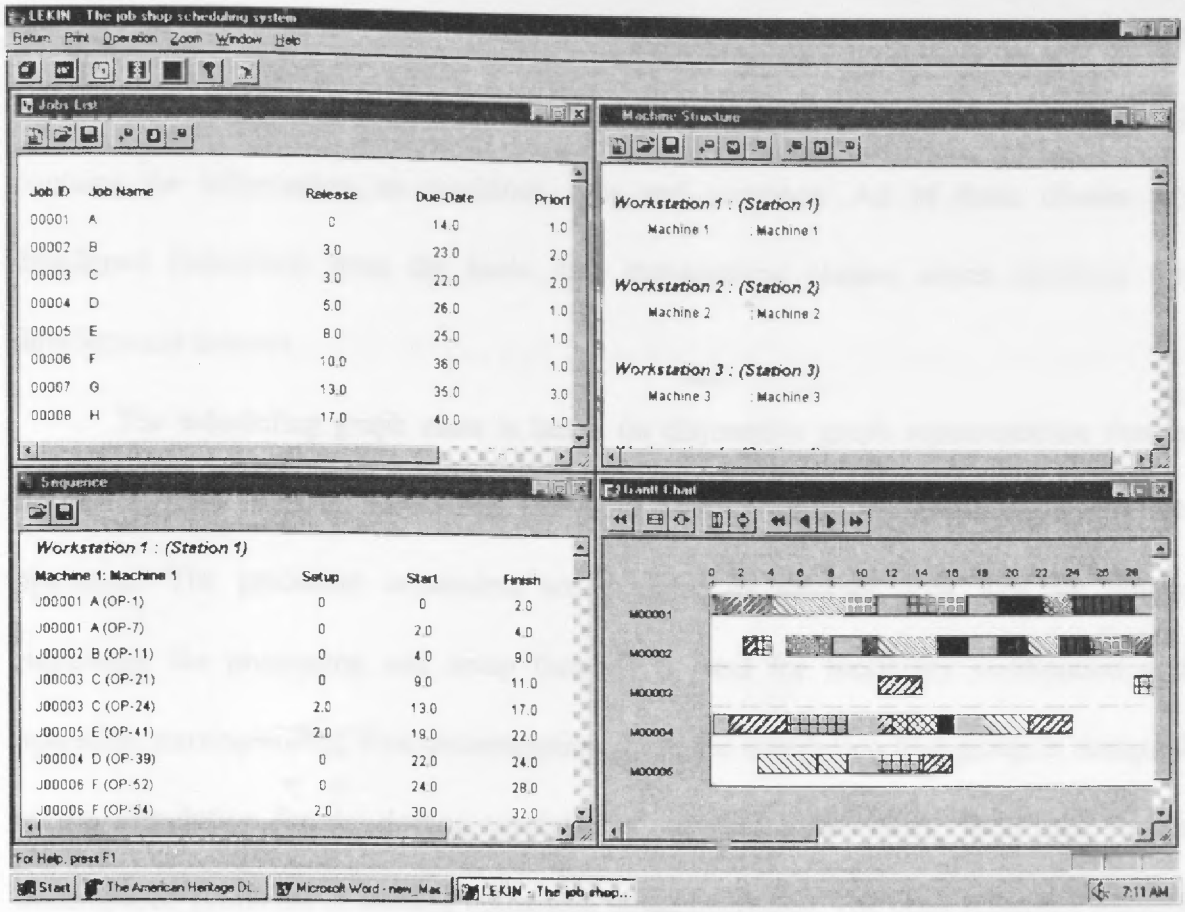


Figure 9-4: Objects relationship

9.4 System Overview

LEKIN is a demonstration of assembly shop scheduling system. The system is designed to ease the researcher on developing new heuristics and testing their performances. Shop schedulers can use it to efficiently manage the scheduling problem. It is based on PC platform operating in Microsoft windows environment (Figure 9-5).


LEKIN The job shop scheduling system
Return Print Operation Zoom Window Help
Jobs List
Machine Structure
Sequence
Gantt Chart
M00001
M00002
M00003
M00004
M00006
For Help, press F1
Start
The American Heritage Di...
Microsoft Word - new_Mac...
LEKIN - The job shop...
7:11 AM

Job ID	Job Name	Release	Due-Date	Priort
00001	A	0	14.0	1.0
00002	B	3.0	23.0	2.0
00003	C	3.0	22.0	2.0
00004	D	5.0	26.0	1.0
00005	E	8.0	25.0	1.0
00006	F	10.0	36.0	1.0
00007	G	13.0	35.0	3.0
00008	H	17.0	40.0	1.0

Workstation 1 : (Station 1)
Machine 1 : Machine 1

Workstation 2 : (Station 2)
Machine 2 : Machine 2

Workstation 3 : (Station 3)
Machine 3 : Machine 3

Workstation 1 : (Station 1)

Machine 1: Machine 1	Setup	Start	Finish
J00001 A (OP-1)	0	0	2.0
J00001 A (OP-7)	0	2.0	4.0
J00002 B (OP-11)	0	4.0	9.0
J00003 C (OP-21)	0	9.0	11.0
J00003 C (OP-24)	2.0	13.0	17.0
J00005 E (OP-41)	2.0	19.0	22.0
J00004 D (OP-39)	0	22.0	24.0
J00006 F (OP-52)	0	24.0	28.0
J00006 F (OP-54)	2.0	30.0	32.0

Figure 9-5: LEKIN -- The assembly shop scheduling system

The system can be divided into two sections -- database & tools and user interface. The database & tools section manages the data and is the host for the heuristics. After the sequence is generated, the information is sent to the user interface section for display in various forms.

9.5 Database and Tools Section (DT)

This section is developed with C++ based on ANSI C++. Therefore, the codes are portable to other systems that have ANSI C++ compiler. The database and tools (DT) are designed based on object oriented methodology. There are four groups of objects (classes) in this section -- basic data management, database, scheduling graph, and scheduling tools. As it is named, the database group is designed for data management. It contains the information on machines, jobs and sequence. All of these classes are developed (inherited) from the basic data management classes which facilitate the development process.

The scheduling graph class is based on disjunctive graph representation that is frequently used in many scheduling heuristics. Each node in the graph represents the operation. The precedent constraints are modeled by directed arcs. The arc length represents the processing and setup time. It is used for feasibility verification and operation starting/ending time determination. Last, the scheduling tool group is designed as the foundation for developing workstation sequencing heuristics. It is used in the decomposed problems in shifting bottleneck based heuristics. The class reference manual can be found in the appendix.

Database

The system has a large internal database. It allows users to work on problems as large as 32,000 operations and 32,000 machines. The database can be divided into sections as follows.

Machine: LEKIN is designed to store machine information as a list of workstations. A workstation is a group of machines performing similar tasks. The machines in the same workstation may not be identical. Some machines may work faster than others. The machine speed is referenced to the average speed of machines in the same workstation. This type of machines are normally referred as *related machines*. Some machines may require setup before processing a particular job. The setup time depends on the current machine status and the new status required to process the coming job. The system provides a lookup table which allows up to ten types of machine status for each machine. In case that some machines in a workstation have restriction that they cannot process some particular tasks, users are able to assign only a partial group of machines in the workstation to process those tasks.

Job: The system stores job information as a list. Each job is another list of operations. The processing route may start from a number of roots. They are assembled to sub-parts and parts are assembled to the final product.

Availability: There are two types of machine available time list kept in the database. First, it is the working shift. Users can set up to six working periods per day. For example, the first production shift is from 9am-12noon and 1pm-5pm. The second shift runs from 5pm-9pm and 9pm-1am. The third shift is from 1am-4am and 5am-9am. These six working periods is for Monday-Friday. The shifts on Saturday and Sunday can be set differently. Second, it is the holidays that the shop will be closed. The holiday can be set as a full day or a partial day.

Heuristics

LEKIN provides basic dispatching rule such as EDD, SPT, LPT, ATCS, etc. and new efficient heuristic like shifting bottleneck heuristics developed in this research. It allows users to customize their heuristics and link them to the system. Furthermore, users may input their own sequences manually.

Disjunctive Graph

The directed graph is developed from the job data. When a sequence is generated, the selections are assigned to the graph. The job completion time as well as the operations starting time can be determined. If the infeasible sequence is entered, the graph can detect it and provide an error message. There are two graphs generated by the system. The first one is a directed graph (no selection). Another one is the active graph (disjunctive graph). The directed graph is used to improve the re-sequencing steps.

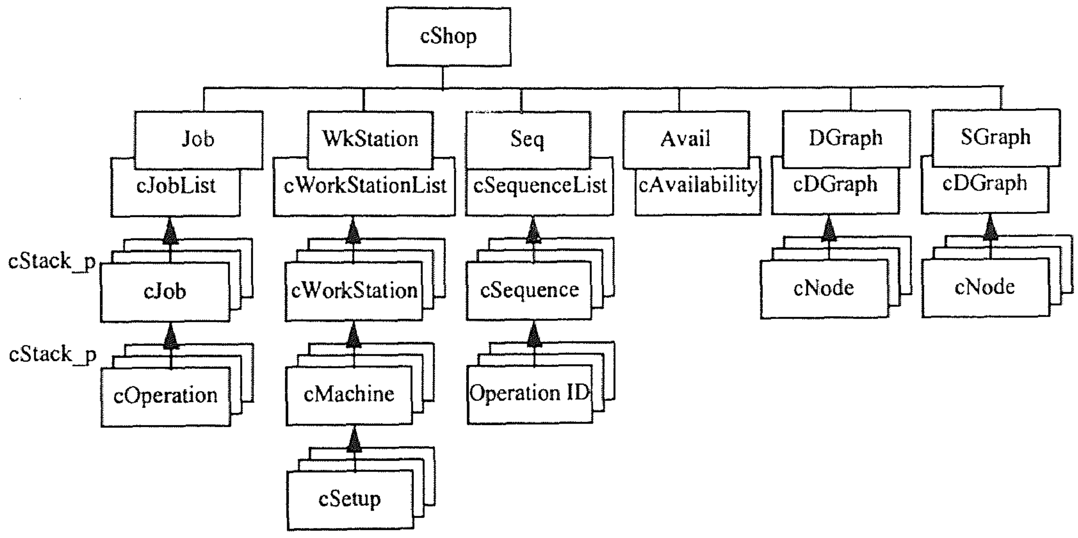


Figure 9-6: Structure of the cShop class

Objects and Classes

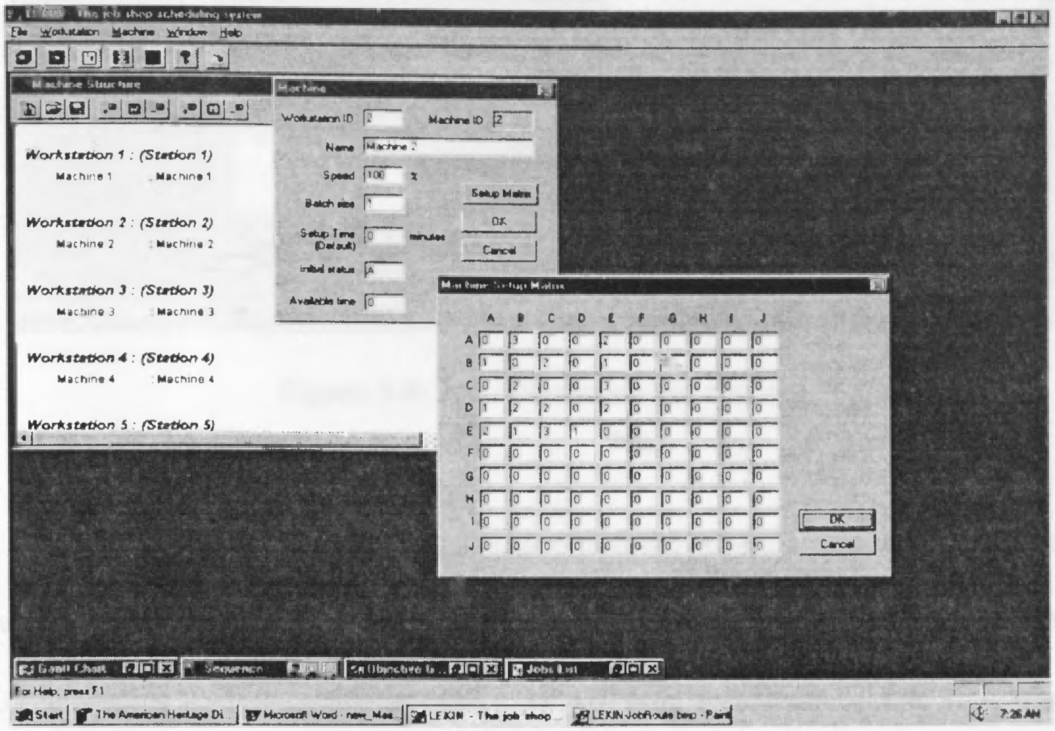
cShop is the main class in this section (Figure 9-6). It contains the necessary information of the shop. There are two members in this class that worth mentioning. DGraph, which is created from cDGraph class, extracts the job information and produces the directed graph. After the sequence is generated, the selections are added to the directed graph. The new graph is stored in SGraph object. The operation starting, completion, processing, and setup time are determined from this object. Furthermore, cShop is the host for various heuristics. It provides general structures for priority rule and shifting bottleneck based heuristics. Only the user defined indexing rule is needed for developing a new priority rule. The provided sub-problem sequencing methods or user defined ones may be used to develop a new shifting bottleneck based heuristics.

9.6 User Interface Section

There are many computer platforms available in the market. Our objective is to develop a powerful scheduling system that can be easily applied in small to medium size machine shops. We select PC based machine as our target due to the fact that most companies already have PC in their production facility.

The system is a 32 bits application designed to work with Microsoft Windows 95 and NT. Windows 3.1 and Windows for Workgroup need to update a component call "win32s" to let them run 32 bits application. The instruction on how to update win32s is provided in the appendix. The codes are developed and compiled using Visual C++. The graphic user interface (GUI) help users on data entering, visualizing and comparing the results while the database & tools section is used as the core on generating the sequence.

When using the system, first, the machine structure information needs to be entered. After the workstation information is input, the corresponding machines can be inserted. The system can handle both single machine or parallel (identical or related) machines workstation. Setup matrix is used when machines require new setting as a result of changing from one status to another (Figure 9-7).



Workstation 1 : (Station 1)
Machine 1 : Machine 1

Workstation 2 : (Station 2)
Machine 2 : Machine 2

Workstation 3 : (Station 3)
Machine 3 : Machine 3

Workstation 4 : (Station 4)
Machine 4 : Machine 4

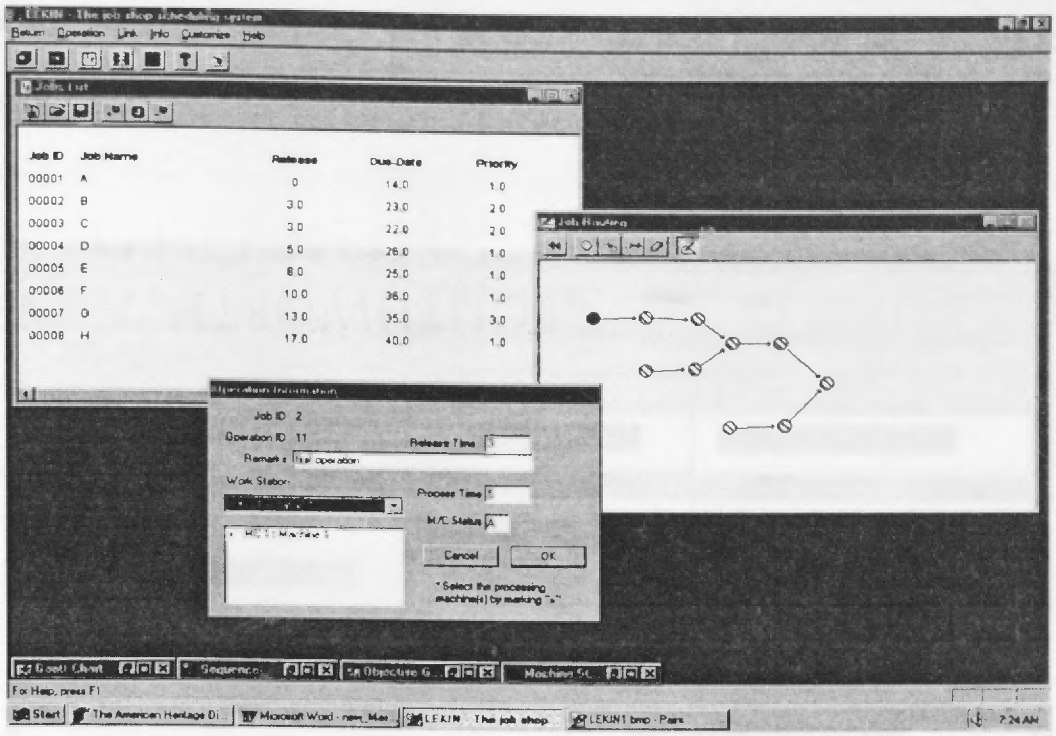
Workstation 5 : (Station 5)

Workstation ID 2 Machine ID 2
Name Machine 2
Speed 100 %
Batch size 1
Setup Time (Default) 0 minutes
Initial status A
Available time 0

	A	B	C	D	E	F	G	H	I	J
A	0	3	0	0	2	0	0	0	0	0
B	1	0	2	0	1	0	[illegible]	0	0	0
C	0	2	0	0	3	0	0	0	0	0
D	1	2	2	0	2	0	0	0	0	0
E	2	1	3	1	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	0

Figure 9-7: Entering machines information

After completing machine structure information, job information can be entered. The job routing and machine assignment are done in the routing window. The route is created from nodes and links. Assembly operation is formed by linking *k* nodes to a node. Each node (operation) must be assigned the processing workstation and a group of machines in that workstation that is able to process it (Figure 9-8).

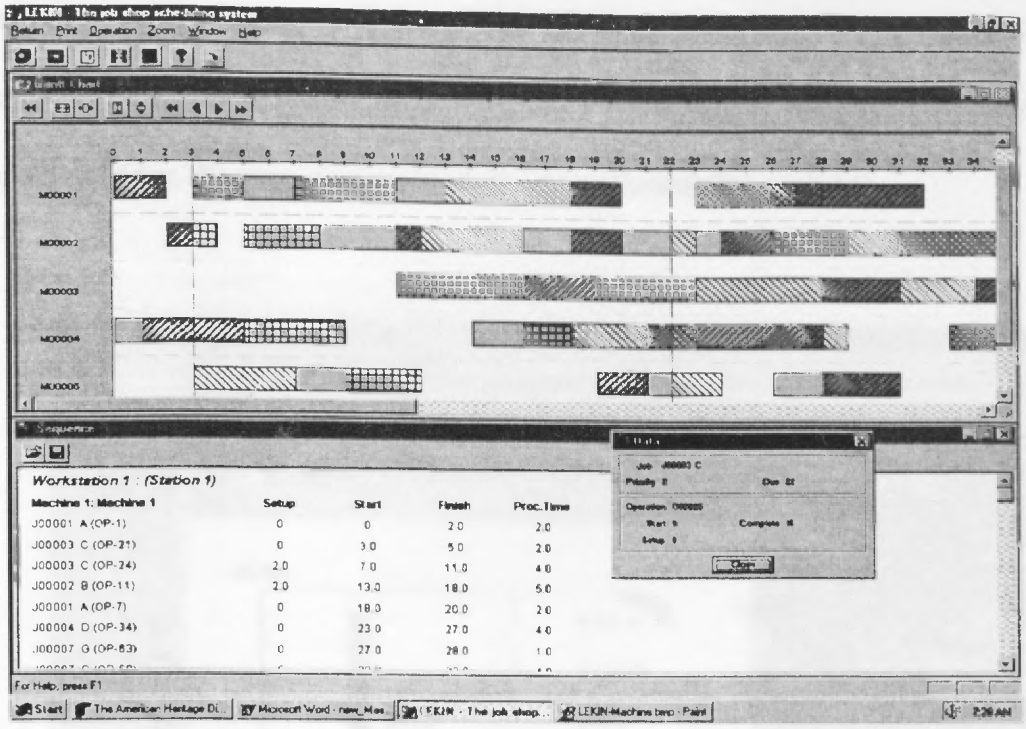


Job ID	Job Name	Release	Due-Date	Priority
00001	A	0	14.0	1.0
00002	B	3.0	23.0	2.0
00003	C	3.0	22.0	2.0
00004	D	5.0	26.0	1.0
00005	E	8.0	25.0	1.0
00006	F	10.0	36.0	1.0
00007	G	13.0	35.0	3.0
00008	H	17.0	40.0	1.0

Job ID 2
Operation ID 11
Release Time [illegible]
Remarks [illegible] operation
Work Station
Process Time [illegible]
M/C Status A
Cancel OK
* Select the processing machine(s) by marking "*"

Figure 9-8: Job information and routing

After all the needed information is specified, the sequence can be generated by the provided heuristics, manual entry, or user defined heuristic. The GUI will send a message to DT activating the appropriate heuristic. DT will send a message back to GUI when the sequence has been generated. The sequence for each machine including its details such as setup duration and starting/completion time of the operations will be displayed in the sequence window along with various shop performance indices. The system is able to display the sequence in graphical form using Gantt chart (Figure 9-9). Users can visualize the sequence. Furthermore, they are able to modify the sequence manually by using the pointing device (mouse). The newly generated sequence is passed to DT to check for the feasibility. The chart will be updated if the sequence is feasible. Normally, the system will generate the semi-active sequence. However, upon request, it can generate non-active sequence where some machines are kept idle though there are some jobs waiting for the processing,.


LEKIN - The job shop scheduling system
Return Print Operation Zoom Window Help
Gantt Chart
M00001
M00002
M00003
M00004
M00005
Sequence
Data
Close
For Help, press F1
Start
The American Heritage Di...
Microsoft Word - new_Mas...
LEKIN - The job shop...
LEKIN-Machine two - Paint

Workstation 1 : (Station 1)

Machine 1: Machine 1	Setup	Start	Finish	Proc.Time
J00001 A (OP-1)	0	0	2.0	2.0
J00003 C (OP-21)	0	3.0	5.0	2.0
J00003 C (OP-24)	2.0	7.0	11.0	4.0
J00002 B (OP-11)	2.0	13.0	18.0	5.0
J00001 A (OP-7)	0	18.0	20.0	2.0
J00004 D (OP-34)	0	23.0	27.0	4.0
J00007 G (OP-63)	0	27.0	28.0	1.0

Figure 9-9: Gantt chart window

The system allows users to attach their own heuristic. The heuristic can be developed with any computer language. However, a scheduling library is provided for C++ users. This library is helpful for developing custom priority rules or custom shifting bottleneck based heuristics. For a quick interface, users may enter the sequence manually.

After a sequence is generated, it may be stored in a log book. The log book works as a temporary memory for the sequence generated. The sequence will be lost if it has not been saved as a file before exiting the system. The sequences stored in it can be retrieved even after the sequence has been cleared and a new sequence has been generated. The log book can hold up to six sequences.

Another use of the log book is that the sequences stored in it can be compared graphically using *objective* command (Figure 9-10). One of the six standard performance indices including makespan, maximum lateness, maximum weighted lateness, weighted tardiness, weighted flow time, and number of late jobs may be selected. Two indices may be compared in the same graph with *multi-obj* command (Figure 9-11).

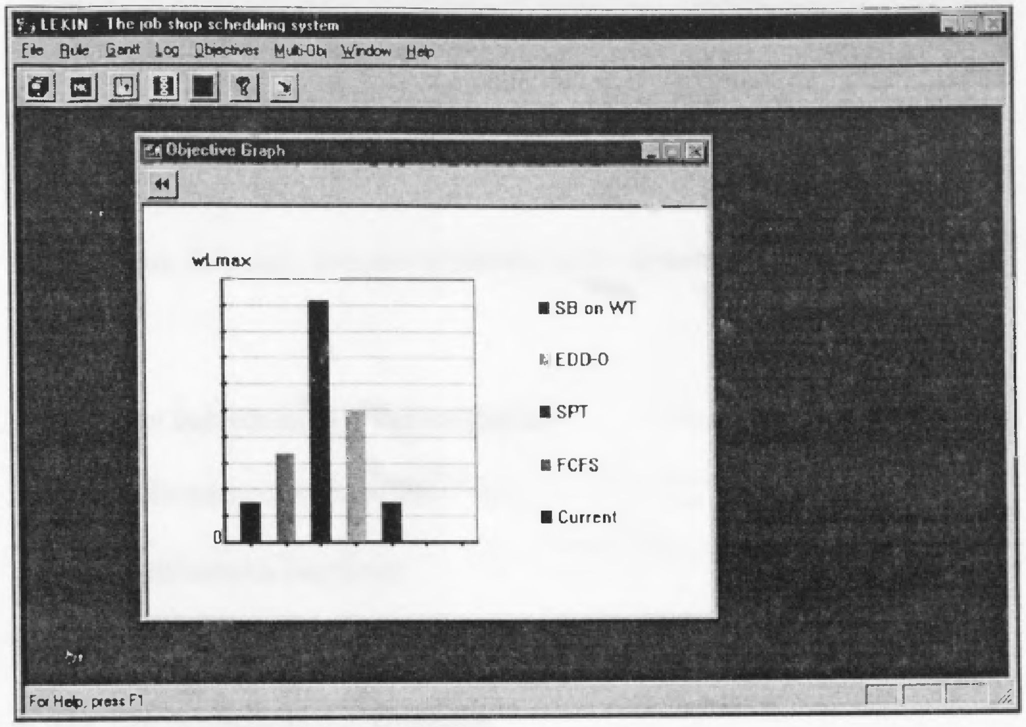

LEKIN - The job shop scheduling system
File Rule Gantt Log Objectives Multi-Obj Window Help
Objective Graph
wLmax
0
■ SB on WT
■ EDD-0
■ SPT
■ FCFS
■ Current
For Help, press F1

Figure 9-10: Comparing objective values among the sequences saved in the log book

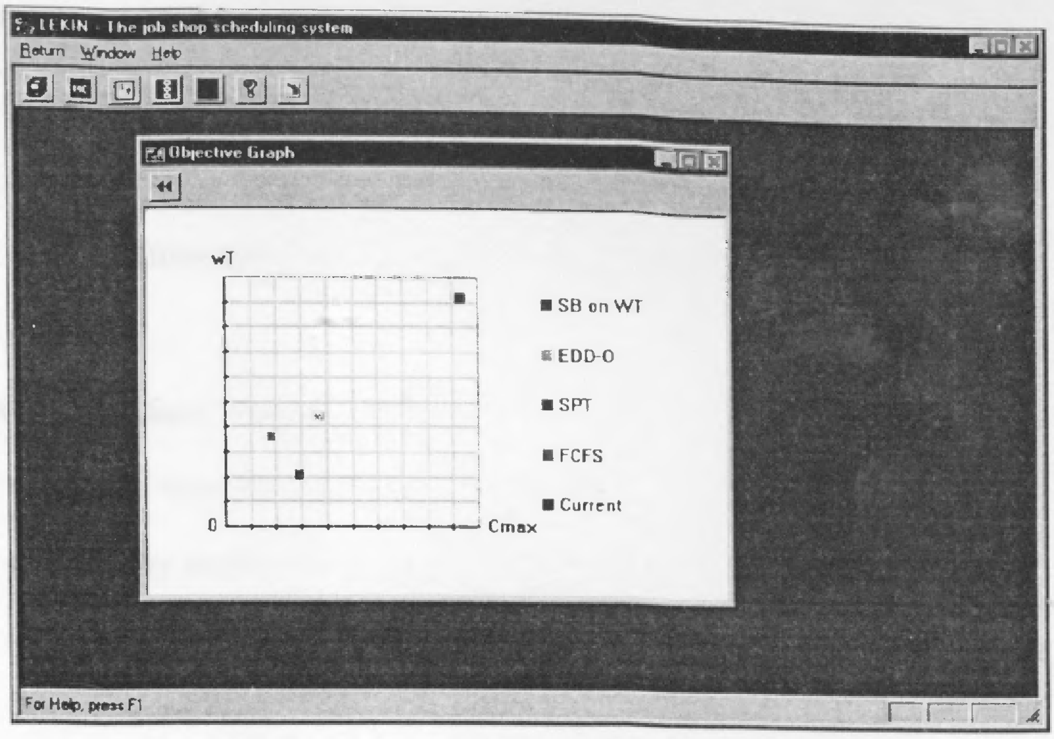


Figure 9-11: Comparing multiple objectives

There are five main screens (windows) in the system as follows.

9.6.1 Machine Information Window (input)

- Add/edit/remove workstations
- Add/edit/remove machines

9.6.2 Job Information Window (input)

- Add/edit/remove/sort job
- Create/edit job routing
- Edit operation information
- Workstation/machine assignment

9.6.3 Sequence Information Window (output)

- Generate the sequences according to the selected method or apply the user defined heuristic
- Store/retrieve previously generated sequences to/from the log book

9.6.4 Gantt Chart Viewer (output)

- Display sequence graphically (Gantt chart)
- Manually modify the sequence with feasibility checking
- Allow semi-active or non-active sequence generation

9.6.5 Objective Viewer (output)

- Compare the shop performance indices among the sequences stored in the log book and the current one

9.7 User Defined Heuristic

As mentioned earlier, users can attach their own algorithm to the system. He/she needs to create an excusable program. The program must be renamed to "user.exe" and placed in the same directory with the main program.

When calling user defined heuristic command in sequence information window, the system will save the machines and operations information in "_user.wkt" and "_user.job". Then, it calls "user.exe" which is the user defined heuristic. After user.exe is completed, it must save the result in "user.seq". The system will load the sequence and update all the windows.

The scheduling library for C++ users is provided. The library is especially helpful for developing priority rules or shifting bottleneck based heuristics. There is a number of useful classes included in the library.

Example 9-1: The following example is a C++ codes for WSPT rule.

```
// user.exe
#include <iostream.h>
#include "lekin.h"      // include the scheduling library

cRuleReturn *IndexWSPT(cShop *pShop,cMCAvail &MCAvail,
cJob &jbX,cDGraph &Graph,double fK1=0,double fK2=0)
{
  /* this function return a pointer to an operation in jbX that has
     highest index and select the machine that can complete the task
     earliest.

     pShop is a pointer point to the database therefore the job, machine
           information can be accessed through this pointer.
     MCAvail store the machine available time
     jbX is a set of operations that have not been sequenced and will
         not create dead locks.
     Graph is the directed graph after adding partial sequence from
           selected operations
     fK1 and fK2 are parameters that pass to the rule. They are not used
                 for WSPT.
     cRuleReturn is a class used for returning values.
  */
  cRuleReturn *pRtn = new cRuleReturn;
  cOperation *popSelect,*popTemp;
  double     fBest=-fBigM,fTemp;
  int i,iN=jbX.Num();
  for(i=1;i<=iN;i++)
  {
    popTemp = jbX.Get(i);
    fTemp   = popTemp->fWeight /popTemp->fProcessTime;
    if(fTemp>fBest)
    {
      fBest = fTemp;
      popSelect = popTemp;
    }
  }
  pRtn->op = popSelect;
  pRtn->mc = FirstFinish(pShop,MCAvail,popSelect,Graph);
  return pRtn;
}


main()
{
      cShop myShop;
      ofstream outfile("_user.seq");
      myShop.LoadWks("_user.wkt");
```

```
        myShop.LoadJob("_user.job");
        myShop.PriorityRule(IndexWSPT);
        outfile<<myShop.Seq;
        outfile.close();
        return 0;
}
```

9.8 User Interface Classes

The object-oriented user interface classes are developed based on Microsoft Foundation Class (MFC). Single document, CAsmDoc, is created and registered in CAsmApp. It is linked to Six frames (CMCFrame, CJobFrame, CSeqFrame, CRouteFrame, CObjFrm, and CGChartFrame) and six view widows (CMCView, CJobView, CSeqView, CRouteView, CObjView, and CGChartView). Each frame has its own view, menu and tool bar. Three main windows (Machine Structure, Job, and Sequence windows) can be directly accessed from the main tool bar. There are 28 dialog boxes associated with commands and more than 70 classes in used. As a normal practice, context sensitive help is included. is can be accessed by F1 key, as well, The help file can be accessed from the main tool bar or the menu.

9.9 Database and Tools Classes

The classes developed in this system can be used as a tool in heuristic development. We divide the classes into four groups as shown below. The detail of each class and it syntax are explained in the "lekin.h".

9.9.1 Database Group Classes

In the data base group, there are three major classes -- job, workstation and sequence. The classes and sub-classes are listed as follows.

- cJobList : list of cJob
  - cJob : list of cOperation
    - cOperation : basic data element of job.
- cWorkstationList : list of cWorkstation
  - cWorkstation : a group of machines performing similar tasks
    - cMachine : basic element of workstation
- cSequenceList : list of cSequence
  - cSequence : list of operation ID that formed into a sequence on a machine
- cShop : the main class that host all the above classes

9.9.2 Basic Data Management Classes

- cStack : linked list and stack for integers
- cStack_f : linked list and stack for doubles
- cStack_l : link list and stack for unsigned longs
- cStack_p : link list and stack for pointers
- cArray_f : array of double that can extend its size automatically
- cArray_l : array of long integer that can extend its size automatically
- cArray_p : array of pointers that can extend its size automatically
- cArray2_f : array size 2 of double that can extend its size automatically
- cArray2_l : array size 2 of long integer that can extend its size automatically
- str : string of characters that can extend its size automatically
- cDate : manipulate the date information

9.9.3 Scheduling Graph Classes

• cNode	: node in the graph
• cArc	: arc in the graph
• cDGraph	: combine list of nodes and arcs to form a graph

9.9.4 Scheduling Tool Classes

• cTool	: based class for other scheduling tool class
• cSMTool	: single machine sequencing tool
• cMMTool	: parallel machine sequencing tool
• cBMTool	: batch machine sequencing tool

CHAPTER 10

CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

We have presented the heuristic development for assembly type job-shop scheduling problems. They are based on decomposition technique called shifting bottleneck. This technique was developed by Adams *et al.* (1988) for job-shop scheduling problem with minimize makespan objective. We modify this technique to the new environments and the new objectives.

In order to develop the heuristic, various sub-problems need to be studied. These sub-problems including single machine, parallel machine, and batch machine sequencing problems are not found in the literature. Various types of techniques are used on developing the heuristics for these sub-problems including greedy method, beam search, critical path analysis, local search and dynamic programming.

The first sub-problem is single machine scheduling with tails. A priority rule called TER is developed. This priority rule outperforms most of the standard priority rules. We develop a critical path improvement (CPI) method that, further, improve the sequence generated from TER. A local search technique is appended to the heuristic when sequence dependent setup is considered.

The second sub-problem is parallel machine scheduling problem with tails. We propose two heuristics. The first one is an extension of the results from single machine scheduling problem with tails. The other technique is based on linear sequence representation and beam search.

The third sub-problem is batch machine scheduling problem with tails. First, we consider the case that the processing time of the operations are equal. A batch allocation technique based on dynamic programming is developed. The heuristic can determine the appropriate batch sizes. Then, we extend the results to multiple families jobs where jobs from different families cannot be processed in the same batch, and dependent setup is concerned.

The proposed heuristic for assembly type job-shops is based on bottleneck concept. It formulates sub-problems dynamically and solve them iteratively to arrive at a satisfactory solution. The test results prove that this technique is far better than current techniques (priority rules) used in practice. We extend the research to other objectives, i.e. minimizing the makespan, minimizing the maximum weighted lateness and minimizing the weighted flow time. Small modifications on the sub-problems are required. As expected, the test results show that the method is superior to the priority rules.

Last, we develop a scheduling system, LEKIN, as the linkage between the theoretical research and the practical world. The system can be separated into two parts. The first part is the database and tools (DT). The other part is the user interface (UI). DT is developed with ANSI C++ based on object oriented design. The classes are placed in the library. They can be used as the tools for developing new heuristics. UI section is developed with Visual C++. It operates on Windows 95 or Windows NT environments. The user interface provides various type of interfaces including text, Gantt chart and various type of graphs. The heuristics developed in this research can be applied.

Future Steps

When an order of multiple items of the same product is released to the shop floor, there are two simple alternatives to model the problem. The order can be considered as multiple jobs or as one huge job. Both methods have some pitfalls. For the first solution, the problem size will increase drastically. We may not be able to track the solution. For the later one, we have restricted the problem size. However, this restriction will degrade the solution as the later processing cannot start until all the items in the order have been completed. Lot sizing tends be the good compromise. The problem will be tractable and the degrading of the solution is subsided. However, a new formulation may be a better alternative. The new disjunctive graph representation needs to be developed.

Next, the machines that we studied in the problems are discrete. Continuous machine that accepts continuous feeds are found in some shops. The processing time of these machines may be referred as the *cycle time*, c_j. *Feed rate*, f_j, indicates the capacity of the machine. After a set of tasks have entered the machine, the first job will leave the machine after c_j. Then, every f_j unit of time, a task will be completed. Although continuous machines are seldom found in the machine shop, it is worth studying.

REFERENCES

1. Aarts, E. H. L. and P. J. M. Van Laarhoven. 1985. "Statistical Cooling: A General Approach to Combinatorial Optimization Problems." *Philips J. Res.* 40: 193-226.

2. Adams, J., E. Balas, and D. Zawack. 1988. "The Shifting Bottleneck Procedure for Job Shop Scheduling." *Management Science*. 34(3): 391-401.

3. Ansari, A. and B. Modarress. 1995. "Wireless Kanban." *Production and Inventory Management Journal*. 36(1): 60-64.

4. Ansari, A. and B. Modarress. 1991. "Transportation Systems for Meeting Just-in-Time Materials Delivery Requirements." *Production Planning and Control*. 2(3): 273-279.

5. Arizona, I., A. Yamamato, and H. Ohto. 1992. "Scheduling for Minimizing Total Actual Flow Time by Neural Networks." *International Journal of Production Research*. 30(3): 503-511.

6. Aytug, H., S. Bhattacharyya, G. J. Koehler, and J. L. Snowdon. 1994. "A Review of Machine Learning in Scheduling." *IEEE Transactions on Engineering Management*. 41(2): 165-171.

7. Balas, E. 1969. "Machine Sequencing Via Disjunctive Graphs: An implicit enumeration algorithm." *Operation Research*. 17: 941-957.

8. Balas, E., J. K. Lenstra, and A. Vazacopoulos. 1995. "The One-Machine Problem with Delayed Precedence Constraints and Its Use in Job Shop Scheduling." *Management Science*. 41(1): 94-109.

9. Barnes, J. W., and J. Chambers. 1992. "Solving the Job Shop Scheduling Problem Using Tabu Search." *Technical Report Series ORP91-06*. Graduate Program in Operations Research, The University of Texas at Austin.

10. Barnes, J. W. and M. Laguna. 1992. "Solving the Multiple-Machine Weighted Flow Time Problem Using Tabu Search." *IIE Transactions*.

11. Blackstone, J. H., D. T. Phillips, and G. L. Hogg. 1982. "The State-of-the-Art Survey of Dispatching Rules for Manufacturing Job Shop Operations." *Int. J. Prod. Res.* 20(1): 27-45.

12. Blazewicz, J., M. Dror, and J. Weglarz. 1991. "Mathematical Programming Formulations for Machine Scheduling: A Survey." *European Journal of Operations Research*. 51: 283-300.

13. Carlier, J. 1982. "The One-Machine Sequencing Problem." *European Journal of Operational Research*. 11: 42-47.

14. Carlier, J. and E. Pinson. 1989. "An Algorithm for Solving the Job Shop Problem." *Management Sci*. 35: 164-176.

15. Castillo, E. D. 1992. "An application of network scheduling optimization in a pharmaceutical firm." *Computers in Industry*. 18: 279-287.

16. Cerny, V. 1985. "Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm." *J. Opt. Theory Appl*. 45: 41-51.

17. Cho, H. and A. Wysk. 1994. "A Robust Adaptive Scheduler for an Intelligent Workstation Controller." *International Journal of Production Research*. 31(4): 771-789.

18. Chryssolouris, G., M. Lee, and M. Domroeese. 1991. "The Use of Neural Networks in Determining Operational Policies for Manufacturing Systems." *Journal of Manufacturing Systems*. 10(2): 166-175.

19. Croce, F. D., R. Tadei, and G. Volta. 1995. "A Genetic Algorithm for The Job Shop Problem." *Computers Ops. Res*. 22(1): 15-24.

20. Dauzere-Peres, S. and J. B. Lasserre. 1993. "A Modified Shifting Bottleneck Procedure for Job-Shop Scheduling." *International Journal of Production Research*. 31(4): 923-932.

21. Dauzere-Peres, S. and J. Paulli. 1994. "Solving the General Multiprocessor Job Shop Scheduling Problem." *Management Report no. 182*. Rotterdam School of Management, Erasmus Universiteit Rotterdam, The Netherlands.

22. Davis, L. (Editor) 1991. *Handbook of Genetic Algorithms*. New York, NY: Van Nostrand Reinhold.

23. Davis, L. 1985. "Job Shop Scheduling with Genetic Algorithms." *Proc. Int. Conf. Genetic Algorithms and Their Applications* (Edited by J. . Grefenstette). Lawrence Erlbaum: 136-140.

24. Dell'Amico, M. and M. Trubian. 1993. "Applying Tabu-Search to the Job-Shop Scheduling Problem." *Annals of Operations Research*. 41: 231-252.

25. Doctor, S R., T. M. Cavalier, and P. J. Egbelu. 1993. "Scheduling for machining and assembly in a job-shop environment." *International Journal of Production Research*. 31(6): 1275-1297.

26. Dorndorf, U. and E. Pesch. 1995. "Evolution Based Learning in a Job Shop Scheduling Environment." *Computers Ops. Res*. 22(1): 25-40.

27. Elorata, E., A. Lehtonen, and K. Tanskanen. 1995. "Fast, Flexible and Cooperative Supply Chains - Key Issues for the Survival of European Industry." *Production, Planning & Control.* 6(3): 238-245.

28. Foo, Y.P.S., and Y. Takefugi. 1988a. "Stochastic Neural Networks for Solving Job-shop Scheduling: Part 1. Problem Presentation." *Proceeding of the ICNN.* 2: 275-282.

29. Foo, Y.P.S., and Y. Takefugi. 1988b. "Stochastic Neural Networks for Solving Job-shop Scheduling: Part 2. Architecture and Simulations." *Proceeding of the ICNN.* 2: 283-290.

30. Foo, Y.P.S., and Y. Takefugi. 1988c. "Integer Linear Programming Neural Networks for Job-Shop Scheduling." *Proceeding of the ICNN.* 2: 341-348.

31. Ford, H. 1926. *Today and Tomorrow.* Garden City, NY: Doubleday Page & Co.

32. Fox, M. S. 1983. *Constraint-Directed Search: A Case Study of Job-Shop Scheduling*. Ph.D. Dissertation, Department of Computer Science , Carnegie-Mellon University, PA.

33. Fox, M. S., and S. F. Smith. 1984. "The role of intelligent reactive processing in production management." *Proc. CAM-I's 13th Ann. Meeting and Technical Conf.* Clearwater Beach, FL: 13-15.

34. Fry, T., J. F. Cox, and J. H. Blackstone. 1992. "An Analysis and Discussion of the Optimized Production Technology Software and Its Use." *Production and Inventory Management.* 1(2): 229-242.

35. Garey, M. R. and D. S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York, NY: W. H. Freeman and Company.

36. Gershwin, S. B. 1991. "Assembly/disassembly systems: an efficient decomposition algorithm for tree-structured networks." *IIE Transactions.* 23: 302-314.

37. Giffler, B. and G. L. Thompson. 1960. "Algorithms for Solving Production Scheduling Problems." *Operations Research.* 8: 487-503.

38. Glover, F. 1990. "Tabu Search - Part II." *ORSA Journal on Computing.* 2(1): 4-32.

39. Glover, F. 1989. "Tabu Search - Part I." *ORSA Journal on Computing.* 1(3): 190-206.

40. Glover, F. 1987. "Tabu Search Methods in Artificial Intelligence and Operations Research." *ORSA Artificial Intelligence Newsletter.* 1(2): 6.

41. Glover, F. 1986. "Future Paths for Integer Programming and Links to Artificial Intelligence." *Computers and Operations Research.* 13(5): 533-549.

42. Glover, F. 1977. "Heuristics for Integer Programming Using Surrogate Constraints." *Decision Sciences.* 8(1): 156-166.

43. Glover, F., J. P. Kelly, and M. Laguna. 1995. "Genetic Algorithms and Tabu Search Hybrids for Optimization." *Computers Ops Res.* 22(1): 111-134.

44. Glover, F., C. McMillan and B. Novick. 1985. "Interactive Decision Software and Computer Graphics for Architectural and Space Planning." *Annals of Operations Research.* 5: 557-573.

45. Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley.

46. Gopalan, M. M. and U. Dinesh Kumar. 1994. "On the transient behavior of a merge production system with *n* end buffer." *Int. J. Prod. Econ.* 34: 157-165.

47. Gopalan, M. M. and U. Dinesh Kumar. 1992. "Stochastic analysis of a two-stage production system with *n* parallel stations in the first stage." *Int. J. Manage. Sys.* 8: 263-275.

48. Hayes, P. V. and S. I. Sayegh. 1992. "A Supervised Neural Network Approach to Optimization As Applied to *n*-Job, *m*-Machine Job Sequencing Problem." *Proceedings of ANNIE'92.*

49. Hilliard, M. R. and G. E. Lipeins. 1988. "Machine Learning Applications to Job Shop Scheduling." *Proc. Workshop on Production Planning and Scheduling.* AAAI-SIGMAN. St Paul, MN.

50. Husbands, P., F. Mill, and S. Warrington. 1991. "Genetic Algorithms, Production Plan Optimization and Scheduling." Proc. 1st Int. Workshop on Parallel Problem Solving from Nature (Edited by H. P. Schwefel and R. Manner). *Lecture Notes Comput. Sci.* 496: 80-84.

51. Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems.* Ann Arbor, Michigan: The University of Michigan Press.

52. Hopfield, J. J. and D. W. Tank. 1985. "Neural Computation of Decisions in Optimization Problems." *Biological Cybernetics.* 52: 141-152.

53. Johnson, M. D. and H. M. Adorf . 1992. "Scheduling with Neural Networks -- The Case Space Hubble Telescope." *Computers and Operations Research.* 19(3/4): 209-252.

[illegible]

54. Kim, S. and Y. Lee. 1993. "Enhancement of a Job Sequencing Rule Using an Artificial Neural Network." *2nd Industrial Engineering Research Conference Proceedings*: 842-846.

55. Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi. 1983. "Optimization by Simulated Annealing." *Science*. 220: 671-680.

56. Kolen, A. W. J. and J. K. Lenstra. 1985. "Combinatorics in Operations Research." *Handbook of Combinatorics*. (Edited by R.L. Graham, M. Grotschel, and L. Lovasz) Amsterdam. North-Holland: 25-28.

57. Laarhovan, P. J. M. V., E. H. L. Aarts and J. K. Lenstra. 1992. "Job Shop Scheduling by Simulated Annealing." *Operation Research*. 40(1): 113-125.

58. Laguna, M., J. W. Barnes and F. Glover. 1991. "Tabu Search Methods for a Single Machine Scheduling Problem." *Journal of Intelligent Manufacturing*. 2: 63-73.

59. Laguna, M., J. W. Barnes and F. Glover. 1992. "Scheduling Jobs with Linear Delay Penalties and Sequence Dependent Setup Costs and Times Using Tabu Search." *Applied Intelligence*.

60. Laguna, M. and J. L. Gonzalez-Velarde. 1991. "A Search Heuristic for Just-in-Time Scheduling in Parallel Machines." *Journal of Intelligent Manufacturing*. 2: 253-260.

61. Lo, Z. P. and B. Bavarian. 1991. "Scheduling With Neural Networks for Flexible Manufacturing Systems." *Proceedings of the IEEE International Conference on Robotics and Automation*: 818-823.

62. Lundy, M. and A. Mees. 1986. "Convergence of an Annealing Algorithm." *Math. Prog.* 34: 111-124.

63. Miller, R. K., E. Lufg, and T.C. Walker. 1988. *Artificial Intelligence Applications in Manufacturing*: 166-177.

64. Monks J. G. 1982. *Operations Management/Theory and Problems*, New York, NY: McGraw-Hill, Inc.

65. Ovacik, I. M., and R. Uzsoy. 1992. "A Shifting Bottleneck Algorithm for Scheduling Semiconductor Testing Operations." *Journal of Electronic Manufacturing*. 2: 119-134.

66. Ow, P. S. and T. E. Morton. 1988. "Filtered Beam Search in Scheduling." *International Journal of Production Research*. 26: 35-62.

67. Papadimtriou, C. H. 1994. *Computational Complexity*. Reading, MA: Addison-Wesley.

68. Pierreval, H. 1993. "Neural Network to Select Dynamic Scheduling Heuristics." *Revue des Sytemes de Decision*. 2(2): 173-190.

69. Pinedo, M. 1995. *Scheduling Theory, Algorithms, and Systems*. Englewood Cliffs, NJ: Prentice Hall.

70. Pinedo, M. and M. Singer. 1995. "First Paper." *Technical paper*, Columbia University, NY.

71. Rabello, L. and S. Alptekin. 1989. "Synergy of Neural Networks and Expert Systems for FMS Scheduling." *Proceedings of The 3th ORSA/TIMS Conference on Flexible Manufacturing Systems: Operations Research Models and Applications*. edited by K. Stecke and R. Suri: 361-366.

72. Rabello, L., A. Jones, and J. Tsai. 1993. "Using Hybrid Systems for FMS Scheduling." *2nd Industrial Engineering Research Conference Proceedings*: 471-475.

73. Reinschmidt, K. F., J. H. Slate, and G. A. Finn. 1990. "Expert systems for plant scheduling using linear programming." *Proc. 4th Int. Conf. Expert Systems in Production and Operations Manage.* Head Island, SC: 198-211.

74. Roman, D B. and A. G. del Valle. 1996. "Dynamic assignation of due-dates in an assembly shop based in simulation." *International Journal of Production Research*. 34(6): 1539-1554.

75. Romeo, F. and A. L. Sangiovanni-Vincentelli. 1985. "Probabilistic Hill Climbing Algorithms: Properties and Applications." *Proceedings 1985 Chapel Hill Conference on VLSI*, Chapel Hill, NC: 393-417.

76. Sabuncuoglu, I. and D. L. Hommertzheim. 1992. *Artificial Neural Networks: Investigations and Developments of Neural Networks for Scheduling Problems*. Orlando, FL: TIMS/ORSA Joint National Meeting.

77. Sadeh, N. and M. S. Fox. 1996. "Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem." *Artificial Intelligence*. 86: 1-41.

78. Sadeh, N., K. Sycara, and Y. Xiong. 1995. "Backtracking techniques for the job shop scheduling constraint satisfaction problem." *Artificial Intelligence*. 76: 455-80.

79. Satake, T., K. Morikawa, and N. Nakamura. 1994. "Neural network approach for minimizing the makespan of the general job-shop." *International Journal of Production Economics*. 33: 67-74.

80. Schonberger, R. 1982. *Japanese Manufacturing Techniques: Nine Hidden Lessons in Simplicity*. New York, NY: The Free Press.

81. Schwefel, H. P. 1977. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*. Birkhauser, Basel.

82. Sebaaly, M. F. and H. Fujimoto. 1996. "Genetic planner for assembly automation." *Proceedings of the IEEE Conference on Evolutionary Computation 1996*. 401-406.

83. Shaw, M. J. P. 1988. "Knowledge-Based Scheduling in Flexible Manufacturing Systems: An Integration of Pattern-Directed Inference and Heuristic Search." *Internatinal Journal of Production Research*. 6: 821-844.

84. Shaw, M. J. P., S. Park, and N. Raman. 1992. "Intelligent Scheduling with Machine Learning Capabilities: The Induction of Scheduling Knowledge." *IIE Transactions on Design and Manufacturing*. 24: 156-168.

85. Shaw, M. J. P. and A. B. Whinston. 1989. "An Artificial Intelligence Approach to the Scheduling of Flexible Manufacturing Systems." *IIE Transactions*. 21: 170-183.

86. Sim, S. K., K. Y. Yeo, and W. H. Lee. 1994. "An Expert Neural System for Dynamic Job Shop Scheduling." *Int. J. Prod. Res.* 32(8): 1759-1773.

87. Simon, J.T. and W.J. Hopp. 1991. "Availability and average inventory of balanced assembly-like flow systems." *IIE Transactions*. 23(2): 161-168.

88. Spencer, M. S. and J. F. Cox. 1995. "MRP in repetitive manufacturing." *International Journal of Production Research*. 33(7): 1881-1899.

89. Spencer, M. S. and J. F. Cox. 1995b. "Master Production Scheduling Development in a Theory of Constraints Environment." *Production and Inventory Management Journal*. 36(1): 8-14.

90. Sprague, R. H., Jr. and E. D. Carlson. 1982. *Building Effective Decision Support Systems*. Englewood Cliffs, NJ: Prentice-Hall Inc.

91. Starkweather, T., D. Whitley, K. Mathias, and S. McDaniel. 1992. "Sequence Scheduling with Genetic Algorithms." *New Directions for Operations Research in Manufacturing* (Edited by G. Fandel, T. Gulledge and J. Jones). Springer, Berlin: 129-148.

92. Taillard, E. 1989. "Parallel Taboo Search Technique for the Job Shop Scheduling Problem." *Research Report ORWP 89/11*. Department de Mathematiques, Ecole Polytechnique Federale de Lausanne.

93. Tang, L. L., Y. Yih, and C. Y. Liu. 1993. "A study on decision rules of a scheduling model in an FMS." *Computers in Industry*. 22: 1-13.

94. Townsend, M. A. and T. W. Lamb. 1991. "Detailed simulation of a real world job shop with subassembly requirements." *Simulation*. 57(2): 114-128.

95. Uzsoy, R., C.Y. Lee and L.A. Martin-Vega. 1992. "A Review of Production Planning and Scheduling Models in the Semiconductor Industry, Part I: System Characteristics, Performance Evaluation and Production Planning." *IIE Transactions*. 24: 47-61.

96. Van Hulle, M. M. 1991a. "Goal Programming Network for Linear Programming." *Biological Cybernetics*. 65: 243-252.

97. Van Hulle, M. M. 1991b. "A Goal Programming Network for Mixed Integer Linear Programming: A Case Study for the Job-shop Scheduling Problem." *International Journal of Neural Systems*. 2(3): 201-209.

98. Van Laarhoven, P., E. Aarts, and J. Lenstra. 1988. "Job Shop Scheduling by Simulated Annealing." *Operations Research*. 40(1): 113-125.

99. Widmer, M. 1991. "Job Shop Scheduling with Tooling Constraints: a Tabu Search Approach." *Journal of the Operations Research Society*. 24(1): 75-82.

100. Widmer, M. and A. Hertz. 1987. "A New Approach for Solving the Flow Sequencing Problem." *ORWP 87/15*. Department of Mathematics, University of Lausanne.

101. Whitley, D., T. Starkweather and D. Fuquay. 1989. "Scheduling Problems and Traveling Salesmen: The Genetic Edge Recombination Operator." *Proc. 3rd Int. Conf. Genetic Algorithms* (Edited by J. D. Schaffer). Morgan Kauffmann:133-140.

102. Woodruff, D. L. and M. L. Spearman. 1992. "Sequencing and Batching for Two Classes of Jobs with Deadlines and Setup Times." *Journal of the Production and Operations Management Society*.

103. Yamada, T. and R. Nakano. 1992. "A Genetic Algorithm Applicable to Large-Scale Job-Shop Problems." *Proc. 2nd Int. Workshop on Parallel Problem Solving From Nature*. (Edited by R. Manner and B. Manderick) : 281-290.

104. Yih, Y. 1990. "Trace-driven Knowledge Acquisition (TDKA) for Rule-Based Real Time Scheduling Systems." *Journal of Intelligent Manufacturing*. 1: 217-230.

105. Yih, Y., T. Liang, and H. Moskowitz. 1993. "Robot Scheduling in a Circuit Board Production Line: A Hybrid ORR/ANN Approach." *IIE Transactions*. 5(2): 26-33.

106. Zhou, D. N., V. Cherkassky, T. R. Baldwin, and D. E. Olson. 1991. "A Neural Network Approach to Job-shop Scheduling." *IEEE Transactions on Neural Networks*. 2(1): 175-179.

107. Zhou, D. N., V. Cherkassky, T. R. Baldwin, and D. W. Hong. 1990. "Scaling Neural Network for a Job-shop Scheduling." *Proceeding of the ICNN*.