Batch Scheduling of Two-machine Limited-buffer Flowshop with Setup and Removal Times

A Thesis
Presented to
The Academic Faculty

By

Jianbin Dai

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Industrial and Systems Engineering

Georgia Institute of Technology
November 2003

Batch Scheduling of Two-machine Limited-buffer Flowshop with Setup and Removal Times

Approved by:

Dr. Chen Zhou, Chairman

Dr. Jane Ammons

Dr. Mark Ferguson

Dr. Spyros Reveliotis

Dr. Martin Savelsbergh

Date Approved 11-21-03

Acknowledgements

First, I would like to take this opportunity to express my sincere appreciation to Dr. Chen Zhou, who is my advisor during this research work, for his unreserved guidance, support, and encouragement. I will always be grateful for his tutoring in all aspects of my life during the past years.

I wish to thank Dr. Jane Ammons, Dr. Mark Ferguson, Dr. Sypros Reveliotis, and Dr. Dr. Martin Savelsbergh for serving on my committee and for advice and constructive comments throughout my Ph.D. program. Especially thank Dr. Ammons for her unreserved instruction on PCB assembly applications.

Thanks must also go to Dr. Hayriye Ayhan and Dr. Anton Kleywegt. They served my first-two-year advisors at Georgia Tech. I really appreciate their guidance in my new life of this country.

I would also like to express my sincere thankfulness to my friends at Georgia Tech in the past years: Junxia Chang, Yonggang Guan, Wei Hua, and Kaiwei Li. I really appreciate their help and friendship.

I cannot overstate the gratitude to my parents and my brothers who encourage and support me every step of the way. Finally, I cannot express with words my deep appreciation and gratitude to Nancy. Her support, patience, encouragement, and sacrifice will forever be appreciated.

Table of Contents

List of Tables

List of Figures

Notation

B_i	Batch i
b_i	Batch size of batch B_i
b_i^*	The steady state batch size for batch B_i
c	The intermediate buffer size
C_i	The makespan for a single batch B_i
$C_{\max}^*$	The optimal makespan of the
C_σ	The makespan of a sequence σ
M_i	Machine i
N	The set of all batches
n	The number of batches
p_{ij}	The processing time of batch B_i on machine M_j
R_{ij}	The removal time of batch B_i on machine M_j
S	The subset of all scheduled batches
S_{ij}	The setup time of batch B_i on machine M_j
$SP(h,i)$	The coupling cost between batch B_h and batch B_i
SP_i^0	The fixed cost of batch B_i
$SP^*(U)$	The optimal makespan for unscheduled batch set U

$SP_i(\sigma)$	The span of batch B_i in a sequence σ
SP_σ	The span of a sequence σ
T_σ	The tail of a sequence σ
U	The subset of all unscheduled batches
σ	A scheduled sequence

Summary

This research investigates the batch scheduling problem for a two-machine limited-buffer flowshop with setup and removal times considered. Different from most previous research, this thesis includes key features of real-life applications. In reality, machine setup is a significant element that cannot be neglected in many situations. Different setup strategies may have different impacts on flowshop scheduling. The intermediate buffer usually has a limited capacity, sometimes rather small. In many cases, similar jobs are grouped together as a family such that no machine setup is required between jobs within the same family. All jobs in the same family must be processed consecutively. Our objective is to find the sequence of jobs on the machines such that the overall completion time is minimized.

In this dissertation, we show that under the steady state conditions, the batch scheduling problem can be converted to a special structured traveling salesman problem (TSP) that can be optimally solved in $O(n\log n)$ time. We develop two approximation algorithms, both of which provide lower bounds for the value of the optimal solution. A branch and bound algorithm is developed to find an optimal solution for the general case. We also perform a case study for a specific electronic assembly application. Finally, various numerical experiments show the effectiveness of our algorithms.

Chapter 1

Introduction

1.1 Background

A *flowshop* consists of a set of machines arranged in series, and a set of jobs visiting machines in a fixed order. At each machine, a specified task is being performed. Each machine can handle only one job at a time. Preemption of operations is not allowed. If all jobs are processed in the same order on each machine, such a schedule is a *permutation* schedule. In a simple conveyorized system, which applies the first-come-first-served (FCFS) principle, a permutation schedule is the only feasible schedule. The problem of flowshop scheduling with *makespan minimization* is to find a sequence of jobs on the machines of the line such that the overall completion time for all jobs is minimized. Makespan minimization is one of the most commonly considered criteria in scheduling literature, and in the context of repetitive assembly, to some extent, makespan minimization is equivalent to cycle time minimization [Crama, 2000]. As we will show in the following chapter, in a two-machine flowshop, makespan minimization is equivalent to minimizing the total idle and blocking times on two machines, and hence maximize the utilization of the machines.

Flowshop scheduling problems have been extensively studied over the past decades. One of the most significant results was Johnson's rule in optimally solving the two-machine unlimited-buffer flowshop scheduling problem with makespan minimization. It has also been shown that two-machine blocking flowshop with makespan minimization can be optimally solved in polynomial time. Unfortunately, most of other flowshop scheduling problems turned out to be *NP*-hard. A large number of heuristic algorithms have been developed in recent years to find approximate solutions of various flowshop scheduling problems. However, only a few of them considered real-life application systems. In reality, machine setup is a significant element that cannot be neglected in many situations. Different setup strategies may have different impacts on flowshop scheduling. The intermediate buffer usually has a limited capacity, sometimes rather small. In many cases, similar jobs are grouped together as a family such that no machine setup is required between jobs within the same family. All jobs in the same family must be processed consecutively. In this thesis, we consider a two-machine flowshop scheduling problem that arises in a specific electronics assembly circumstance.

1.2 Problem Definition

Consider a flowshop with two machines M_1 and M_2 and an intermediate storage buffer of capacity c, that is, the buffer can host at most c parts. There are n batches of jobs, B_1,

B_2, ..., B_n to be processed. All batches are available at time zero. Each batch B_i consists of b_i identical jobs. Each job in batch B_i requires processing times p_{i1} and p_{i2} on machines M_1 and M_2 respectively. These processing times are usually calculated by specific software. We assume that all processing times are given, and do not depend on batch sequence. All jobs from the same batch must be processed consecutively, that is, batches cannot be split. If machine M_1 completes processing on a job and the buffer is full, then the part keeps M_1 blocked until there is room in the buffer. A setup time S_{ij} is required to install component feeder into machine M_j ($j = 1,2$) before a batch B_i being processed on that machine, and a removal time R_{hj} is required to unload the feeder from machine M_j after a batch B_h completes its processing. Setup time S_{ij} and removal time R_{ij} are only determined by batch B_i. The setup of batch B_i on machine M_2 can be done before completion of the operation of the first job of batch B_i on machine M_1, if there exits some idle time on machine M_2. Our objective is to minimize the makespan of all batches. A formal definition of the problem is as follows:

Definition 1.1 (Batch Scheduling Problem) *Given a two-machine flowshop with an intermediate buffer of capacity c and a set of n batches B_1, B_2, ..., B_n, (batch B_i of size b_i and job processing times p_{i1} and p_{i2} on two machines respectively, for i = 1, 2, ..., n) requiring sequence independent setup times (S_{i1} and S_{i2}) and removal times (R_{i1} and R_{i2}) on both machines, we want to find a sequence σ of all batches so that the makespan is minimized.*

1.3 Previous Work

As in many textbooks, we use a triplet $\alpha / \beta / \gamma$ to describe a scheduling problem. The α field describes the machine environment and contains a single entry. Typical machine environments specified in the α field include: single machine (1), parallel machines (P_m), flowshop (F_m), open shop (O_m), and job shop (J_m). The β field provides details of processing characteristics and contains zero, single or multiple entries. Possible entries in the β field are: release dates (r_j), preemptions (*prmp*), precedence constraints (*prec*), blocking (*block*), no-wait (*no-wait*) and so on. The third field γ contains the objectives to be optimized and contains one or multiple entries. Common objectives to be minimized include: makespan (C_{max}), maximum lateness (L_{max}), total weighted completion time ($\sum w_j C_j$), and total weighted tardiness ($\sum w_j T_j$). In the following literature review, we focus on related flowshop scheduling problems.

Flowshop scheduling problems have been extensively studied for decades. In particular, makespan minimization is the objective that has received significant attention in literature. Two-machine infinite-buffer flowshop with makespan minimization ($F_2 // C_{\max}$) was optimally solved by Johnson [1954] in $O(n \log n)$ time. However, it has been shown that $F_m // C_{\max}$ is strongly *NP*-hard when $m \geq 3$ [Gary et al., 1976]. Williamson and Hoogeveen [1997] proved that unless $NP = P$, there is no polynomial time approximation algorithm with a worst-case performance guarantee better than 5/4 for the problem $F_m // C_{\max}$ with an arbitrary number of machines.

Two-machine flowshop with infinite buffer and sequence-independent setup times was first solved by Yoshida and Hitomi [1979]. Sule [1982] extended the problem with both sequence-independent setup and removal times. The problem of two-machine flowshop with batch setups was shown to be *NP*-hard if batches are allowed to be split [Kleinau, 1993]. However, if the number of batches is fixed and processing orders on the two machines are constrained to be identical, the problem is polynomial solvable by dynamic programming [Potts and Wassenhove, 1997].

The aforementioned research work of the flowshop scheduling problem assumes that the buffer capacity between machines is unlimited. However, in many real-life systems, the intermediate storage between machines is limited, or even relatively small. Blocking occurs when the intermediate buffer is full and the upstream machine cannot to release a job into the buffer after its processing. There are two makespan minimization scheduling problems closely related to the limited buffer flowshop: the *blocking* flowshop scheduling problem ($F_m / block / C_{\max}$) and *no-wait* flowshop scheduling problem ($F_m / no-wait / C_{\max}$). In a blocking flowshop, there is no storage between machines. Actually, any limited buffer flowshop can be formulated as a blocking flowshop since each storage space capable of containing one job may be regarded as a machine on which the processing times of all jobs are equal to zero. In a no-wait flowshop, there is also no storage between machines. Differing from blocking, a job when it goes through the system is not allowed to wait at any machine. That is, whenever it has completed its processing on one machine, the next machine has to be idle so that the job does not have to wait. In contrast to the blocking case, where jobs are pushed down the line by machines upstream that have completed their processing, in a no-wait flowshop, jobs are pulled down the line by machines that have become idle. When a flowshop only consists of two machines, if we release all jobs to the first machine as late as possible while at the same time, keep the makespan unchanged, the blocking flowshop turns into a no-wait flowshop, that is, the problem $F_2 / block / C_{\max}$ is equivalent to $F_2 / no-wait / C_{\max}$.

Both $F_2 / block / C_{\max}$ and $F_2 / no-wait / C_{\max}$ problems can be formulated as a special structured traveling salesman problem (TSP), which was optimally solved by Gilmore and Gomory [1964] in $O(n \log n)$ time. However, when $m \geq 3$, $F_m / block / C_{\max}$ cannot be described as a TSP and has been shown to be strongly *NP*-hard. In contrast, $F_m / no-wait / C_{\max}$ can always be formulated as a TSP, but this TSP is known to be strongly *NP*-hard when $m \geq 3$ [Pinedo, 1995].

The problem becomes extremely difficult when both the batch setups and limited buffer are considered. Only a few of research papers have been found to address on this topic. Agnetis et al. [1997] have proved that batch scheduling of a two-machine flowshop with limited buffer is strongly *NP*-hard even without considering setup times. They also showed that the problem is solvable only when all batches satisfy a steady state condition, which will be explained later. However, they did not give out a practical solution to solve a general case. Several heuristic approaches have been presented in recent years. Bloat [1997] implemented a dynamic program to minimize total blocking time. Leisten [1990] presented a method that seeks to keep the intermediate buffers filled in order to avoid starving the following machines. The heuristic method of tabu search has been used in

[Noman, 1999], [Weng, 2000] and [Nowicki, 1999]. Their results showed that this local search strategy is rather effective in some cases.

To our knowledge, our problem of the two-machine limited-buffer flowshop scheduling with sequence-independent setup and removal times has not been extensively explored in the previous literature.

1.4 Organization

This dissertation is organized as follows: In Chapter 2, we are going to show that under some circumstances, the Batch Scheduling Problem is equivalent to $F_2 / block / C_{\max}$, which can be optimally solved by the Gilmore & Gomory algorithm [Pinedo, 1995]. In Chapter 3, we present two approximation approaches, and both of them provide lower bounds of the optimal solution. In Chapter 4, we are going to detail a branch and bound algorithm that optimally solves the problem in theory. In Chapter 5, we apply our algorithm on a specific electronic assembly scheduling problem. Chapter 6 provides some numerical experiment results to show the effectiveness of our approach. Finally, in Chapter 7, we develop some conclusions from our research.

Chapter 2

Steady State Optimization

2.1 Outline

Generally, flowshop batch scheduling problem with limited buffers is *NP*-hard even without considering batch setups and removals. However, Agnetis et al. [1997] have shown that the problem is solvable if all batches satisfy a certain condition, that is, all batches can reach the *steady state*, which will be defined later. Since the term of "steady state" was first defined by Agnetis, we still use this term throughout our dissertation without confusion. In this chapter, we are going to extend the problem with both setup and removal times considered. We find that the problem can still be formulated as a TSP under the corresponding steady state condition and this special structured TSP be can be solved in $O(n \log n)$ time.

2.2 Makespan Evaluation

In this section we are going to show that, for a given feasible sequence σ of all batches, the value of the makespan can be expressed as the sum of two terms if all batches can reach the steady state. The first term of the makespan is a fixed cost, which does not depend on the sequence of the batches. The second term is a coupling cost, which depends on the batch sequence σ; in fact, this is the part we want to minimize. In this section, we assume that $b_i \geq c+1$ for all $i = 1, 2,\ldots, n$. This condition is easily satisfied in our real applications since the buffer capacity usually is much smaller than a batch size. This condition implies that at any time at most two batches are being processed on the two machines.

Agnetis identified that, for a batch B_i with $p_{i1} < p_{i2}$, if the batch size is large enough, blocking on machine M_1 happens when the intermediate buffer is full. A new job is released into the system only when a job completes its processing on machine M_2 and leaves the system. From that point on, batch B_i reaches the steady state as shown in Figure 2.1 (a). On the contrary, if $p_{i1} \geq p_{i2}$, starving on machine M_2 occurs when the intermediate buffer is empty. A new job is released into the system only when a job completes its processing on machine M_1 and enters machine M_2. From that point on, batch B_i also reaches the steady state as shown in Figure 2.1 (b). Then the steady state condition is defined as following.

Definition 2.1 (Steady State) *A batch B_i reaches the steady state at time t if, from then on, the two machines start processing a new part at the same time, and this occurs every*

$max(p_{i1}, p_{i2})$ time units as shown in Figure 2.1. In other words, after time t, the schedule of the batch repeats identically over time.

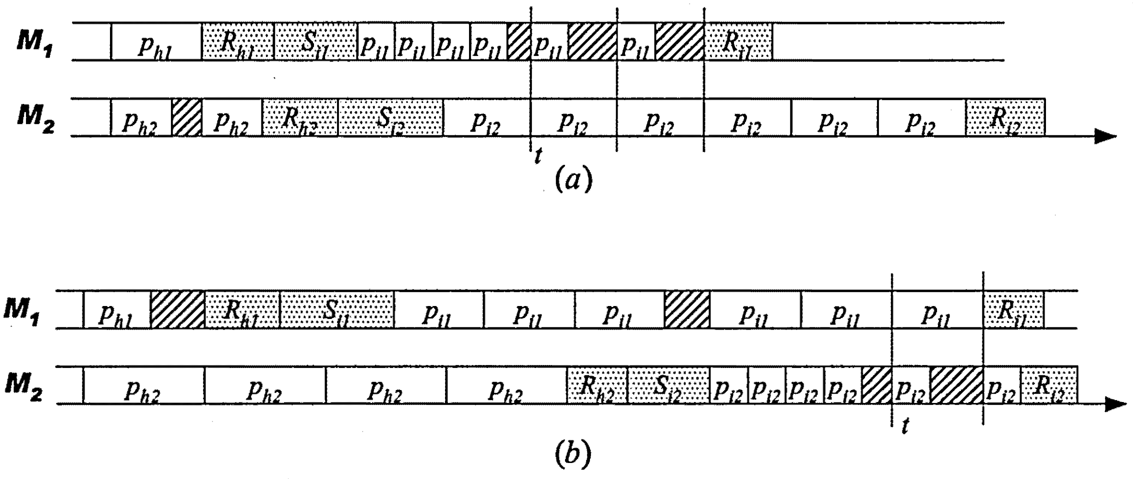


Figure 2.1 The steady state, (a) $p_{i1} < p_{i2}$, (b) $p_{i1} \geq p_{i2}$

Now we define another term that will be frequently used throughout this thesis.

Definition 2.2 (Span) *Given a sequence σ, the span of a batch B_i in σ is defined as the time interval that machine M_1 is busy with batch B_i. And the span of the sequence σ is the sum of spans of all batches in sequence σ.*

In other words, the span of B_i equals the time length from the beginning time of the initial setup of B_i on machine M_1, denoted as time t_1, till the ending time of removal of B_i on machine M_1, which is also called *cleaned-up* time, and we denote it as time t_3, as shown in Figure 2.2. In general, this quantity depends on the sequence σ, and hence we denote the span of B_i as $SP_i(\sigma)$, then $SP_i(\sigma) = t_3 - t_1$. The span of the sequence σ, SP_σ, is calculated by

$$SP_\sigma = \sum_{B_i \in \sigma} SP_i(\sigma) \tag{2.1}$$

Suppose that batch B_i is scheduled immediately following some other batch B_h, and both batches reach the steady states. In the following, we calculate $SP_i(\sigma)$ in the following four possible cases.

1) $p_{i1} < p_{i2}$, $p_{h1} < p_{h2}$

In this case, we define time t_2 as the starting time of batch B_i being processed on machine M_2. Since we have $p_{i1} < p_{i2}$, the second machine M_2 is never idle after time t_2, as shown in Figure 2.2. The value of $(t_3 - t_2)$ equals the total processing time of the first $(b_i - c - 1)$ jobs plus batch B_i's removal time R_{i1} on the first machine M_1, that is, $t_3 - t_2 = (b_i - c - 1)p_{i2} + R_{i1}$.

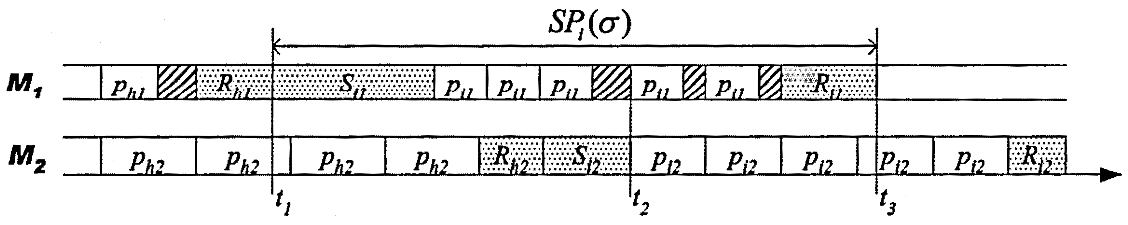


Figure 2.2 $SP_i(\sigma)$ evaluation when $p_{i1} < p_{i2}$, $p_{h1} < p_{h2}$

The time length $(t_2 - t_1)$ equals the maximum of $(c+1)p_{h2} + R_{h2} + S_{i2} - R_{h1}$ and $S_{i1} + p_{i1}$. Therefore,

$SP_i(\sigma) = t_3 - t_1$

$\qquad = (t_2 - t_1) + (t_3 - t_2)$

$$= (b_i - c - 1)p_{i2} + R_{i1} + \max\{S_{i1} + p_{i1}, (c+1)p_{h2} + R_{h2} + S_{i2} - R_{h1}\}$$

$$= (b_i - c - 1)p_{i2} + S_{i2} + R_{i1} + \max\{p_{i1} + S_{i1} - S_{i2}, (c+1)p_{h2} + R_{h2} - R_{h1}\} \tag{2.2}$$

2) $p_{i1} < p_{i2}$, $p_{h1} \geq p_{h2}$

In this case, we define time t_2 as the starting time of batch B_i being processed on machine M_2 just as it is in the previous case. Machine M_2 keeps busy after time t_2, and again $t_3 - t_2 = (b_i - c - 1)p_{i2} + R_{i1}$. The buffer is empty at time t_1, then $(t_2 - t_1)$ equals the maximum of $S_{i1} + p_{i1}$ and $p_{h2} + R_{h2} + S_{i2} - R_{h1}$ as shown in Figure 2.3. Therefore,

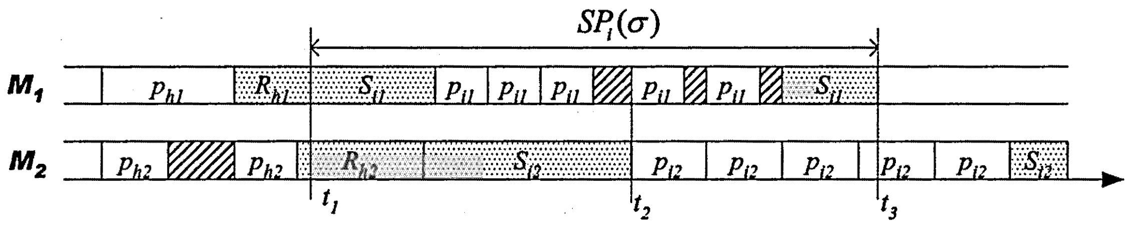


Figure 2.3 $SP_i(\sigma)$ evaluation when $p_{i1} < p_{i2}, p_{h1} \geq p_{h2}$

$$SP_i(\sigma) = (t_2 - t_1) + (t_3 - t_2)$$

$$= (b_i - c - 1)p_{i2} + R_{i1} + \max\{S_{i1} + p_{i1}, p_{h2} + R_{h2} + S_{i2} - R_{h1}\}$$

$$= (b_i - c - 1)p_{i2} + S_{i2} + R_{i1} + \max\{p_{i1} + S_{i1} - S_{i2}, p_{h2} + R_{h2} - R_{h1}\} \tag{2.3}$$

3) $p_{i1} \geq p_{i2}$, $p_{h1} < p_{h2}$

In this case, we define time t_2 as the starting time of the $(c+1)th$ job of batch B_i being processed on machine M_1 as shown in Figure 2.4. Then machine M_1 is never idle after time t_2, and $t_3 - t_2 = (b_i - c - 1)p_{i1} + R_{i1}$. The time value of $(t_2 - t_1)$ equals the maximum of $(c+1)p_{h2} + R_{h2} + S_{i2} - R_{h1}$ and $S_{i1} + (c+1)p_{i1}$. Therefore,

$$\begin{aligned} SP_i(\sigma) &= (t_2 - t_1) + (t_3 - t_2) \\ &= (b_i - c - 1)p_{i1} + R_{i1} + \max\{S_{i1} + (c+1)p_{i1}, (c+1)p_{h2} + R_{h2} + S_{i2} - R_{h1}\} \\ &= (b_i - c - 1)p_{i1} + S_{i2} + R_{i1} \\ &\quad + \max\{(c+1)p_{i1} + S_{i1} - S_{i2}, (c+1)p_{h2} + R_{h2} - R_{h1}\} \end{aligned} \tag{2.4}$$

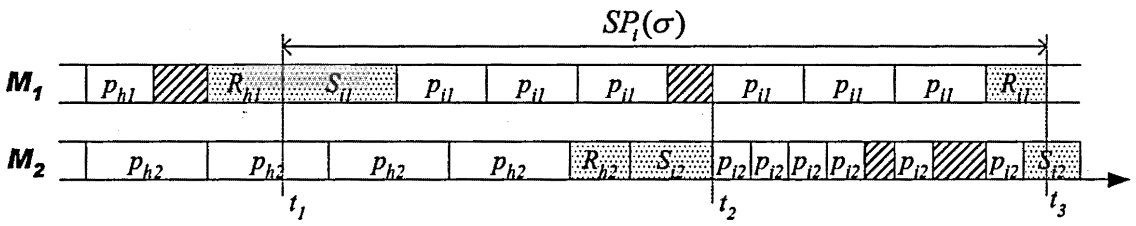


Figure 2.4 $SP_i(\sigma)$ evaluation when $p_{i1} \geq p_{i2}$, $p_{h1} < p_{h2}$

4) $p_{i1} \geq p_{i2}$, $p_{h1} \geq p_{h2}$

In this case, the time t_2 is still defined as the starting time of the $(c+1)th$ part of batch B_i being processed on machine M_1 just as it is in the previous case. Then machine M_1 is never idle after time t_2, and $t_3 - t_2 = (b_i - c - 1)p_{i1} + R_{i1}$. The time length $(t_2 - t_1)$ equals the maximum of $S_{i1} + (c+1)p_{i1}$ and $p_{h2} + R_{h2} + S_{i2} - R_{h1}$ as shown in Figure 2.5. Then,

$$\begin{aligned} SP_i(\sigma) &= (t_2 - t_1) + (t_3 - t_2) \\ &= (b_i - c - 1)p_{i1} + R_{i1} + \max\{S_{i1} + (c+1)p_{i1}, p_{h2} + R_{h2} + S_{i2} - R_{h1}\} \\ &= (b_i - c - 1)p_{i1} + S_{i2} + R_{i1} + \max\{(c+1)p_{i1} + S_{i1} - S_{i2}, p_{h2} + R_{h2} - R_{h1}\} \end{aligned} \tag{2.5}$$

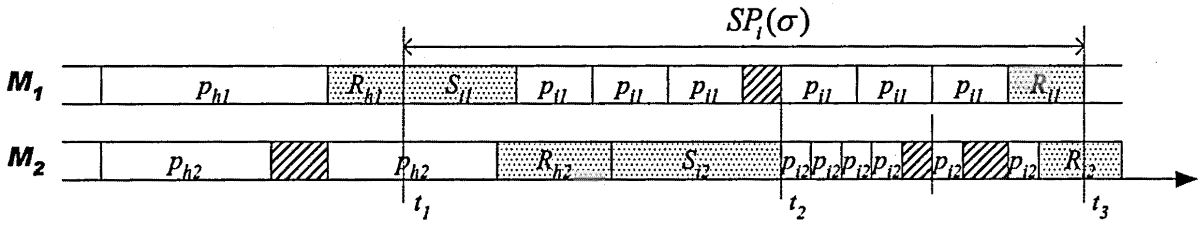


Figure 2.5 $SP_i(\sigma)$ evaluation when $p_{i1} \geq p_{i2}, p_{h1} \geq p_{h2}$

Based on the above four cases, we found that the value of $SP_i(\sigma)$ is composed of two terms. The first term is a fixed cost for batch B_i, which does not depend on the sequence σ, and we denote it as SP_i^0, that is, for the case of $p_{i1} < p_{i2}$,

$$SP_i^0 = (b_i - c - 1)p_{i2} + S_{i2} + R_{i1} \tag{2.6}$$

and for the case of $p_{i1} \geq p_{i2}$,

$$SP_i^0 = (b_i - c - 1)p_{i1} + S_{i2} + R_{i1} \tag{2.7}$$

The second term is a coupling cost, which depends on both batch B_i and the previous batch B_h, but not any other batches in the sequence. We define the following two coupling cost values:

$$E_i = p_{i1}(1 + c\delta(p_{i1} - p_{i2})) + S_{i1} - S_{i2} \tag{2.8}$$

$$F_i = p_{i2}(1 + c\delta(p_{i2} - p_{i1})) + R_{i2} - R_{i1} \tag{2.9}$$

where the function $\delta(x) = 1$ if $x \geq 0$ and $\delta(x) = 0$ if $x < 0$.

Then we can generalize the calculation of $SP_i(\sigma)$ in the following equation:

$$SP_i(\sigma) = SP_i^0 + \max\{F_h, E_i\} \tag{2.10}$$

In the above equation, the value F_h only depends on batch B_h, which is the batch immediately preceding batch B_i, and the value E_i is determined by batch B_i itself. We denote the coupling cost as

$$SP(h,i) = \max\{F_h, E_i\} \tag{2.11}$$

The above discussions can be concluded in the following theorem.

Theorem 2.1 *Suppose that batch B_h immediately precedes batch B_i in a feasible solution σ. If both B_h and B_i reach the steady state, then the span of batch B_i is given by*

$$SP_i(\sigma) = SP_i^0 + SP(h,i) \tag{2.12}$$

where SP_i^0 is a fixed cost of batch B_i, and $SP(h,i)$ is a coupling cost between batch B_i and B_h. The value of $SP_i(\sigma)$ does not depend on the other batches.

2.3 Converting to a TSP

In this section, we are going to show that if all batches can reach the steady state in any feasible sequence σ, the calculation of makespan can be converted to a special structured travelling salesman problem (TSP). And for that particular TSP, an optimal solution in $O(n \log n)$ solving time is available.

Suppose that σ is a feasible sequence of all batches, and in that sequence, all batches reach the steady state. The makespan C_σ equals the sum of all the spans $SP_i(\sigma)$ plus a *tail* T_σ of the whole schedule, as shown in Figure 2.6. For a feasible schedule σ, the tail

T_σ is the time elapsed from the cleaned-up time of the last scheduled batch $B_{\sigma(n)}$ on machine M_1 to the cleaned-up time on machine M_2. Notice that, in some cases, when the removal time on the first machine is much longer than that on the second machine, the value of tail T_σ can be negative as shown in Figure 2.7.

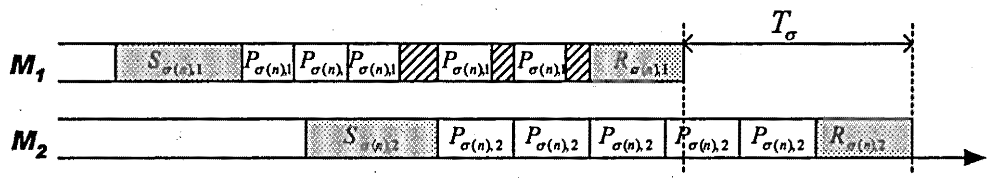


Figure 2.6 The tail of the schedule σ

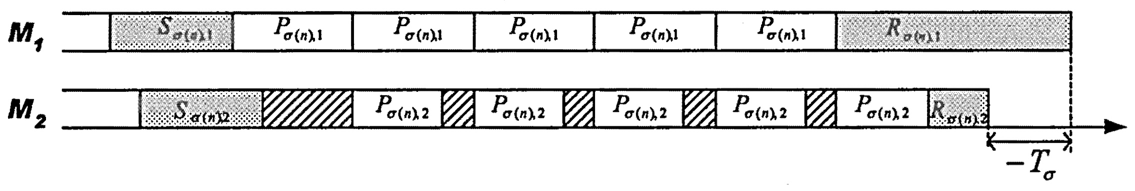


Figure 2.7 Negative-length tail

Then we have the following makespan calculation:

$$C_\sigma = SP_\sigma + \max(0, T_\sigma) = \sum_{i=1}^{n} SP_i(\sigma) + \max(0, T_\sigma) \tag{2.13}$$

In order to calculate the length of the tail T_σ, we regard it as the span of a dummy batch B_{n+1}, consisting of $b_{i+1} = c + 1$ jobs having zero processing times and zero setup and removal times on both machines. Then the value of $\max(0, T_\sigma)$ is exactly the same as the time the dummy batch blocked on M_1. We can rewrite the makespan in following equation:

$$C_\sigma = \sum_{i=1}^{n+1} SP_i(\sigma) = \sum_{i=1}^{n+1} SP_i^0 + \sum_{i=0}^{n} SP(\sigma(i), \sigma(i+1)) \qquad (2.14)$$

In the above equation, the value of SP_{n+1}^0 equals zero, and the batch of $\sigma(0)$ is the dummy batch B_{n+1}. Since the first part of the equation is a constant, the optimal makespan is given by:

$$C_{\max}^* = \min_\sigma \left\{ \sum_{i=1}^{n+1} SP_i(\sigma) \right\} = \sum_{i=1}^{n} SP_i^0 + \min_\sigma \left\{ \sum_{i=0}^{n} SP(\sigma(i), \sigma(i+1)) \right\} \qquad (2.15)$$

Now we consider a TSP with $(n+1)$ nodes. Node 1, 2, ..., n stands for batch B_1, B_2, ..., B_n, respectively, and the distance from node i to node j is equal to $SP(i,j) = \max(F_i, E_j)$, for any $1 \le i, j \le n$. Node $n+1$ stands for the dummy batch. The distance from node $n+1$ to any other node j $(1 \le j \le n)$ is $SP(n+1, j) = E_j$, and the distance from node i $(1 \le i \le n)$ to node $n+1$ is $SP(i, n+1) = F_i$. Then to find a shortest cyclic path linking all $n+1$ nodes is equivalent to minimize the second part of the equation (2.15). In particular, notice that the batch scheduled first in the sequence is the one following the dummy batch in the cyclic path, and the batch scheduled last is the one preceding the dummy batch in the cyclic path. We conclude the above discussions in the following theorem.

Theorem 2.2 *If all batches reach the steady state in any feasible sequence σ, the makespan minimization of Batch Scheduling Problem is equivalent to a TSP with $n+1$*

nodes, in which the distance from node i to node j is given by $SP(i,j) = max(F_i, E_j)$, $1 \le i,j \le n+1$. *The optimal makespan* $C^*_{\max}$ *equals the total length of the minimal cyclic path of the TSP plus a constant* $\sum_{i=1}^{n} SP_i^0$.

Generally, a TSP problem is still hard to solve. However in our case, from equation (2.11), we find that the cost function becomes identical to that of a two-machine blocking Flowshop. This special case of TSP has been solved optimally by Gilmore and Gomory's algorithm in time $O(n \log n)$ [Gilmore and Gomory, 1964]. In the remaining part of this section, we are going to explore this case.

Consider the $F_2 / block / C_{\max}$ problem with two machines in series and zero intermediate storage in between. Notice that in this flowshop, whenever a job starts its processing on the first machine M_1, the preceding job starts to be processed on machine M_2. Then the time length that job j spends on machine M_1, in process or blocked, is $\max(p_{j,1}, p_{j-1,2})$, as illustrated in Figure 2.8.

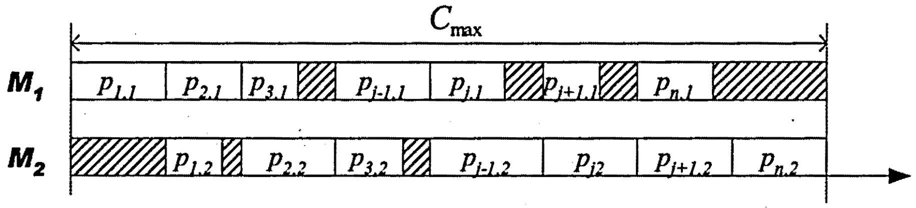


Figure 2.8 A two-machine blocking flowshop

The makespan $C_{\max}$ equals the total time of all jobs spending on machine M_1, either in process or blocked, plus the processing time of the last job on machine M_2, that is

$$C_{\max} = \sum_{j=1}^{n} \max(p_{j,1}, p_{j-1,2}) + p_{n,2} \tag{2.16}$$

where $p_{0,2} = 0$. Then this two-machine blocking flowshop with makespan minimization problem is equivalent to a TSP with $n+1$ nodes, with the distances between every two nodes defined as following:

$$d_{0j} = p_{j1}$$

$$d_{i0} = p_{i2}$$

$$d_{ij} = \max(p_{j1}, p_{i2})$$

This TSP is exactly the same in the form as the one we defined for the Batch Scheduling Problem. Therefore, the Batch Scheduling Problem, when all batches reach the steady state, is equivalent to $F_2 / block / C_{\max}$.

Furthermore, as we can see in Figure 2.8, the following equation holds:

$$2C_{\max} = \sum_{i=1}^{n} (p_{i1} + p_{i2}) \;+\; idle \;\; times \tag{2.17}$$

Since the processing times are fixed constants, makespan minimization problem is equivalent to minimize the sum of all idle times. This result can be generalized in the following statement:

Proposition 2.3 *In a flowshop scheduling problem, makespan minimization is equivalent to minimizing the total idle time.*

The idle time on one of the machines when job j starts on machine M_2 and job k starts on machine M_1 is $|p_{j2} - p_{k1}|$. If $p_{j2} \geq p_{k1}$, job k will be blocked for that time difference on machine M_1, and else if $p_{j2} < p_{k1}$, machine M_2 will remain idle for that time difference. Hence, minimizing the sum of all idle times is equivalent to the following TSP with $n+1$ nodes;

$$d_{0j} = p_{j1}$$

$$d_{i0} = p_{i2}$$

$$d_{ij} = |p_{j1} - p_{i2}|$$

For this particular structured TSP, Gilmore and Gomory's algorithm solves it optimally in time $O(n \log n)$ [Gilmore and Gomory, 1964]. For the details of the algorithm, please refer to Appendix B.

2.4 Sufficient Conditions for the Steady State

As we have seen in the previous sections, the steady state is the key condition for the results in section 2.2. In this section, we are going to develop a sufficient condition under which batch B_i is guaranteed to reach the steady state. In the following, B_h still denotes the batch immediately preceding batch B_i.

Theorem 2.4 *A batch B_i, with $p_{i1} \neq p_{i2}$, reaches the steady state in any feasible sequence σ, if*

$$b_i \geq b_i^* = \lceil c \times max(p_{i1}, p_{i2}) / |p_{i1} - p_{i2}| \rceil + 1 \tag{2.18}$$

Proof. We prove the theorem in the following two cases.

1) $p_{i1} < p_{i2}$

In this case, the steady state is reached only when the buffer is filled up to its capacity. Hence the tail of the previous batch B_h helps batch B_i to reach the steady state. The "worst" case arises when the batch B_h leaves the buffer empty as shown in Figure 2.9.

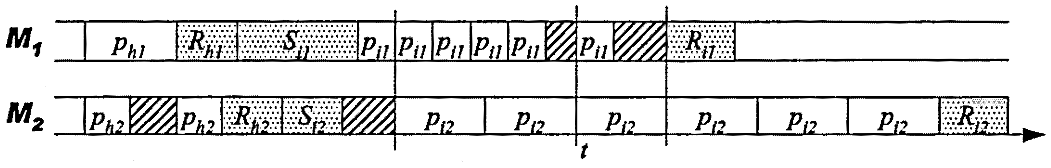


Figure 2.9 The steady state of batch B_i ($p_{i1} < p_{i2}$) in the "worst" case

The steady state is reached when any job is blocked on the first machine. Then we have

$$(b_i - 1)p_{i1} \leq (b_i - (c+1))p_{i2},$$

that is,

$$b_i \geq cp_{i2}/(p_{i2} - p_{i1}) + 1$$

which is in accordance with (2.15). If the batch B_h leaves the buffer not empty, the batch B_i can reach the steady state even sooner.

2) $p_{i1} > p_{i2}$

In this case, the steady state is reached only when the buffer is completely empty. This happens when, for the first time, the part completed on M_1 can be moved to the machine M_2 without waiting in the buffer. The tail of the previous batch B_h slows down B_i to reach the steady state. The worst case arises when the batch B_h leaves the buffer full as shown in Figure 2.10. The steady state is reached when any job is idle on the second machine. Then we have

$$(b_i - 1)p_{i2} \le (b_i - (c+1))p_{i1}$$

that is,

$$b_i \ge cp_{i1}/(p_{i1} - p_{i2}) + 1$$

which is in accordance with (2.15). If the batch B_h leaves the buffer not full, the batch B_i can reach the steady state even sooner. □

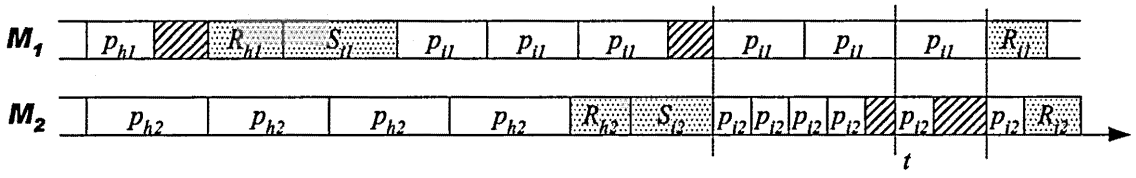


Figure 2.10 The steady state of batch B_i ($p_{i1} > p_{i2}$) in the "worst" case

When $p_{i1} = p_{i2}$, whether the batch B_i can reach the steady state depends on the tail of the previous batch B_h. As shown in Figure 2.11 (a), if the setup on machine M_2 is finished earlier than the first job of batch B_i completes its processing on machine M_1, batch B_i always reaches the steady state. Otherwise, it never reaches the steady state no matter how large the batch size is, as shown in Figure 2.11 (b).

Note that the setup times do not appear in the equation (2.15). This should be not too surprising because the number of parts needed to enter in the steady state in the worst case is not affected by the setup times. Also note that equation (2.15) only gives the sufficient condition to guarantee the steady state, but not a necessary one. In many cases, batches still can reach the steady state even though the condition of (2.18) is not satisfied.

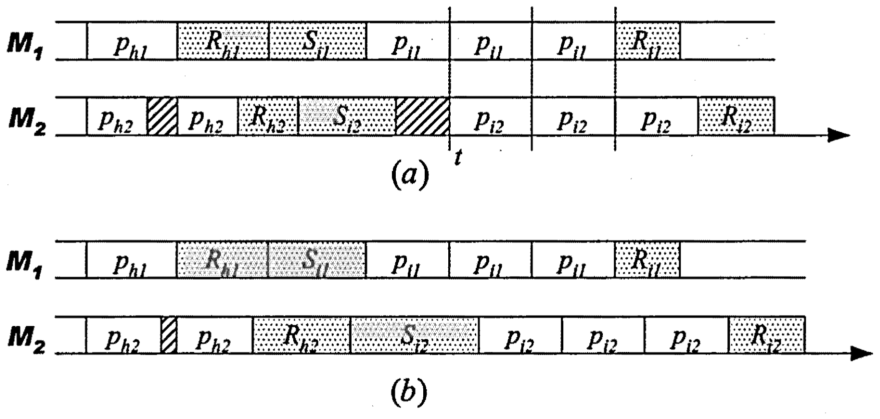


Figure 2.11 The steady state of batch B_l ($p_{l1} = p_{l2}$)

Chapter 3

Lower Bounds and Upper Bounds

3.1 Outline

In Chapter 2 we have proven that the Batch Scheduling Problem, if all batches reach the steady state in any feasible sequencing σ, is equivalent to the two-machine blocking flowshop with makespan minimization problem, which can be optimally solved in $O(n\log n)$ time. However, from equation (2.18) we can see that the steady state condition can be a stringent one when a batch has close processing times on the two machines.

In this chapter, we are going to solve the batch scheduling problem in the following two special circumstances:

1) The processing times on two machines are of much difference for each batch. From equation (2.18) we can see that the threshold value of the batch size is relatively small in this case, and a batch is more likely to reach the steady state. This is easy to

understand since, the difference of the processing times on two machines tends to induce blocking on either machine, which causes the batch to enter the steady state.

2) On the other hand, if all batches have rather close processing times and setup times on both machines, blocking is not likely to happen, which implies the capacity of the intermediate buffer is no longer a constraint to our problem. Consequentially, we can assume that the buffer capacity is unlimited. We are going to provide an optimal solution to this case.

Based on these two points, we are going to design two approximation algorithms, both of which provide lower bounds of the optimal solution. We also provide two upper bounds at the end of this chapter.

3.2 Processing Time Reduction Lower Bound

For convenience, we write here the sufficient condition of the steady state for batch B_i again.

$$b_i \geq b_i^* = \left\lceil cmax(p_{i1}, p_{i2}) / |p_{i1} - p_{i2}| \right\rceil + 1 \tag{3.1}$$

If for some batch, the actual batch size b_i is smaller than b_i^*, we reduce the smaller processing time between p_{i1} and p_{i2} of an amount sufficient to meet the condition (3.1). For an instance, suppose $b_i < b_i^*$, and $p_{i1} \leq p_{i2}$. We set $p'_{i1} = p_{i2} - \lceil cp_{i2}/(b_i - 1) \rceil$. We can always guarantee that the new processing time p'_{i1} is non-negative since we have the assumption that $b_i \geq c+1$. After reducing the processing times, the condition of (3.1) is satisfied for all batches. Then we can apply the approach described in the previous chapter, and get an optimal solution for this processing time reduced problem. Since the modified problem has reduced processing times, this solution is a lower bound of the original Batch Scheduling Problem. The detailed algorithm is as follows:

Algorithm 3.1 (Time Reduction Algorithm)

```
begin
    for i: = 1 to n do
    begin
        $b_i^* := \lceil c\max(p_{i1}, p_{i2}) / |p_{i1} - p_{i2}| \rceil + 1;$
        if $b_i < b_i^*$ then
            if $p_{i1} \le p_{i2}$ then
                $p'_{i1} := p_{i2} - \lceil cp_{i2}/(b_i - 1) \rceil;$
            else $p'_{i2} := p_{i1} - \lceil cp_{i1}/(b_i - 1) \rceil;$
    end;
    solve the $F_2 / block / C_{max};$
end;
```

3.3 Infinite Buffer Lower Bound

In this section, we assume that the intermediate buffer capacity is unlimited. With this assumption, machine M_1 is never idle until all jobs have been processed on it. Suppose that the minimal makespan of this unlimited buffer problem is $C'_{\max}$, and the optimal solution of the original Batch Scheduling Problem is $C^*_{\max}$, then we have $C'_{\max} \le C^*_{\max}$ since we have removed a constraint of the original problem; that is, if we can find an optimal solution of this problem, it provides a lower bound of the original problem.

For a two-machine flowshop without setup times, the makespan minimization problem was first solved by Johnson [1954]. In our problem, each batch has to be processed consecutively. However, this constraint does not bring too much difficulty to the problem since, according to Johnson's rule, all jobs in the same batch are scheduled consecutively. But when the setup and removal times are considered, the problem becomes much more complicated. We are going to develop an algorithm to optimally solve this problem in $O(n \log n)$ time.

We consider a single batch B_i scheduled at time zero as shown in Figure 3.1. Denote C_i the optimal makespan for this single batch scheduling, then

$$C_i = \max(S_{i1} + R_{i2} + C_i^0, S_{i1} + R_{i1} + b_i p_{i1}, S_{i2} + R_{i2} + b_i p_{i2}) \qquad (3.2)$$

where C_i^0 is the optimal makespan of the batch B_i without setup and removal times, that is,

$$C_i^0 = \begin{cases} p_{i1} + b_i p_{i2} & \text{if } p_{i1} < p_{i2} \\ b_i p_{i1} + p_{i2} & \text{if } p_{i1} \ge p_{i2} \end{cases} \qquad (3.3)$$

If we schedule jobs on the second machine M_2 as late as possible while keeping the position of last scheduled job in batch B_i unchanged, as shown in Figure 3.2, the resulting schedule is still feasible and optimal. In Figure 3.2, α_i, β_i and γ_i are defined as following:

$$\alpha_i = C_i - (S_{i2} + R_{i2} + b_i p_{i2}) \quad (3.4)$$

$$\beta_i = C_i - (S_{i1} + R_{i1} + b_i p_{i1}) \quad (3.5)$$

$$\gamma_i = C_i - \alpha_i - \beta_i \quad (3.6)$$

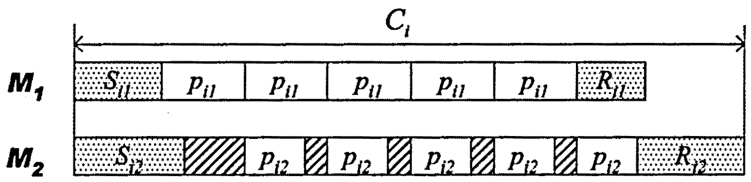


Figure 3.1 Single batch scheduling

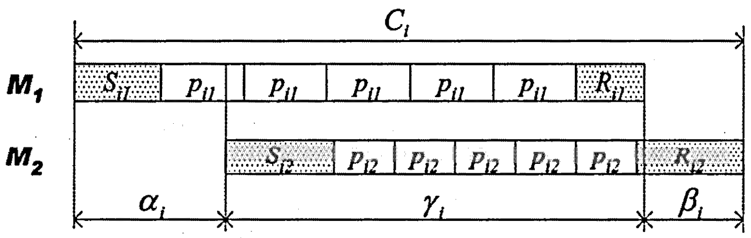


Figure 3.2 Single batch scheduling when jobs are scheduled as late as possible

In this delayed schedule, machine M_2 is idle for the first α_i time units, and machine M_1 is idle for the last β_i time units. Both machines are simultaneously busy for γ_i time units. When all batches are considered, we need to arrange the α_i's and β_i's so that the total idle time is minimized, but we can do nothing on the γ_i portion. If you regard each batch B_i as a single job J_i with processing times α_i on machine M_1 and β_i on machine M_2, This problem is equivalent to the typical two-machine flowshop scheduling problem, which can be optimally solved by Johnson's rule. We provide the unlimited buffer algorithm as follows.

Algorithm 3.2 (Infinite Buffer Algorithm)

```
begin
    for i: = 1 to n do
    begin
        calculate $C_i$ as in equation (3.2);
        calculate $\alpha_i$ and $\beta_i$ as in equations (3.4) and (3.5) respectively;
        define a job with processing time $\alpha_i$ on machine one, and processing time $\beta_i$
        on machine two;
    end;
    solve the $F_2 // C_{max}$ with the job set $\{(\alpha_i,\beta_i), i = 1,2,...,n\}$ by Johnson's rule, and
    get an optimal sequencing $\sigma$;
    schedule the batches according to the sequencing $\sigma$;
end;
```

Theorem 3.1 *Algorithm 3.2 is optimal for the batch scheduling problem of two-machine unlimited-buffer flowshop with setup and removal times.*

Proof. First, we denote $C^*_{\max}$ the optimal makespan of the batch scheduling of two-machine unlimited-buffer flowshop with setup and removal times. For each batch B_i, we create two jobs: J_{i1} with processing times (α_i, β_i), and J_{i2} with processing times (γ_i, γ_i). α_i, β_i and γ_i are defined in equations (3.4) – (3.6). Now we have a typical $F_2 // C_{\max}$ problem with *2n* jobs to be scheduled: $\{J_{11}, J_{12}, J_{21}, J_{22}, ..., J_{n1}, J_{n2}\}$. Suppose the optimal makespan for this new problem is $C'_{\max}$, then we have.

However, since each job J_{i2} $(i = 1,2,...,n)$ has the same processing times on two machines, all these jobs can be scheduled anytime and will not change other batches' states. The optimal sequence for the job set $\{J_{11} J_{21}, ..., J_{n1}\}$ is achieved by Johnson's rule. Now we insert each job J_{i2} $(i = 1,2,...,n)$ into the schedule as follows. On machine M_1, the load of γ_i is scheduled right after α_i, and on machine M_2, the load of γ_i is scheduled right before β_i, as illustrated in Figure 3.3. Then the resulting schedule is feasible and exactly the same as Algorithm 3.2 described. Then we have $C'_{\max} \geq C^*_{\max}$.

Based on the above two points, we have $C'_{\max} = C^*_{\max}$, then Algorithm 3.2 is optimal.□

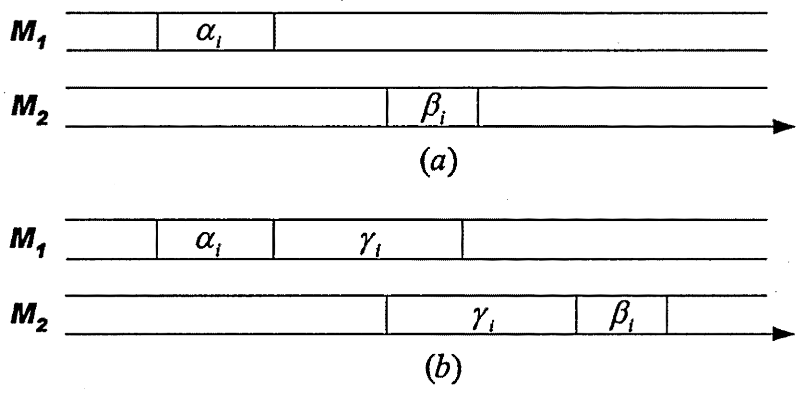


Figure 3.3 Insertion of job J_{i2} into the schedule

3.4 Upper Bounds

Any feasible sequence provides an upper bound for the optimal solution of the batch scheduling problem. In our research, we have two approaches to find upper bounds.

1) Although the approach described in Chapter 2 does not guarantee an optimal solution if some batches cannot reach the steady state condition at any feasible sequences, it still provides a feasible solution. When most of batches can reach the steady state, this approach provides a good upper bound, as will be illustrated in Chapter 5.

2) If we assume that the intermediate buffer capacity is unlimited, we can apply the infinite buffer algorithm, and get another upper bound. This approach is a good upper bound when most batches have close processing times on two machines. This will also be illustrated in Chapter 5.

Chapter 4

A Branch and Bound Algorithm

4.1 Definitions and the Fathoming Rules

In this chapter, we are going to develop a branch and bound algorithm that implicitly enumerates all possible sequences for the batch scheduling problem, in which not all batches can reach the steady state. At each step of the enumeration tree, S is a subset of all batches which have been assigned to the first positions in the schedule. We denote the partial schedule of S as σ. U is the subset of all batches which have not been assigned in the schedule, that is, $U = N - S$, where N is the set of all batches. Then, each node of the enumeration tree is associated with a pair (σ, U).

As in Chapter 2, cleaned-up time is defined as the time point at which a batch finishes its removal on a machine. As illustrated in Figure 4.1, t_1 and t_2 are cleaned-up times of the partial schedule σ on machine M_1 and M_2 respectively. The span of the partial schedule σ, denoted as SP_σ, is the time lapse from time zero to the cleaned-up time of the last batch of σ on machine M_1, that is, $SP_\sigma = t_1$ as shown in Figure 4.1. We also define $T_\sigma = t_2 - t_1$ the tail of the partial schedule σ. Notice that the value of T_σ can be negative when the removal time on M_1 is much longer than that on M_2 as shown in Figure 4.1 (b).

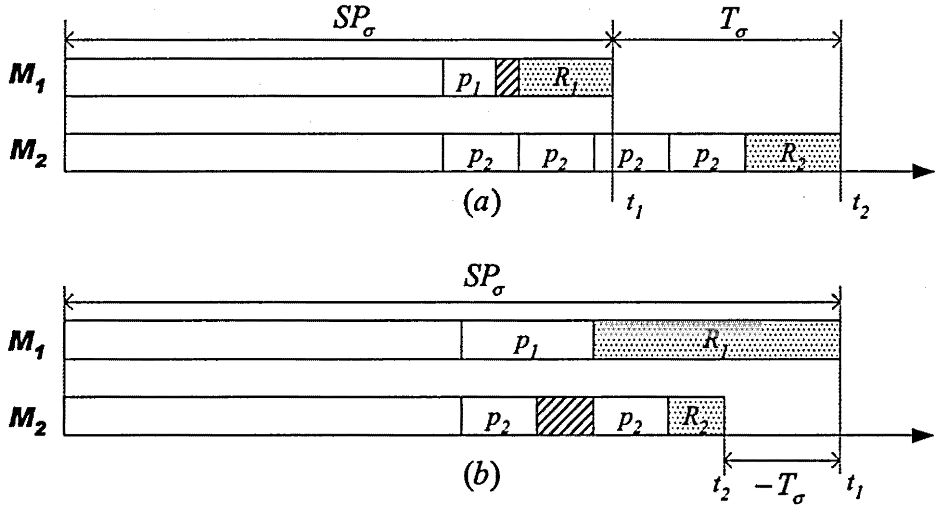


Figure 4.1 Span and tail of a partial schedule, (a) $T_\sigma > 0$, (b) $T_\sigma < 0$

For the set of unscheduled batches U, we denote its minimum makespan as $SP^*(U)$, which is the minimum time length from t_1 to the end of entire schedule. Notice that since there is a tail T_σ left from the scheduled batches, which may be positive or negative, the value of $SP^*(U)$ can be different from the optimal makespan of U when only the unscheduled bathes U are considered. Although not all batches can reach the steady state in our problem, it is still possible that batches in the unscheduled batch set U can reach the steady state. In this case, we can apply the optimal approach described in Chapter 2 to find the value of $SP^*(U)$. Otherwise, we need to calculate a lower bound for $SP^*(U)$.

After these definitions, we design the following fathoming rules for the branch and bound algorithm:

1) We use the Depth-First Search (DFS) algorithm to form an enumeration tree. DFS performs a deep probe, creating a path as long as possible, and backs up one node to initiate a new probe when it can mark no new node from the tip of the path. One of the advantages of the DFS is that we can always get a feasible solution, most of time, a rather good solution, even when the number of nodes to be searched is extraordinarily large. For details of Depth-First Search algorithm, please refer to the reference [Ahuja et al., 1993].

2) At node (σ, U), if the value of $SP^*(U)$ can be calculated exactly, no need to further from this node. If $SP^*(U)$ plus SP_σ is less than the best known upper bound, the best known upper bound is updated by this value. Otherwise, this node can be eliminated.

3) At node (σ, U), if the best available lower bound of $SP^*(U)$ plus SP_σ exceeds the best known upper bound, this node cannot be optimal, and hence can be eliminated.

4) At node (σ, U), if an upper bound of $SP^*(U)$ plus SP_σ is less than the best-known upper bound, the best-known upper bound is updated.

The detailed algorithm will be presented in section 4.5.

4.2 Updating SP_σ and T_σ

In our branch and bound algorithm, we need to calculate SP_σ and T_σ in a large number of times. A straightforward way to calculate SP_σ and T_σ is to build the Gantt chart of the partial schedule σ. However, this approach is not efficient since the computation time is proportional to the batch size. Fortunately, in the branch and bound algorithm, we do not have to compute the span and tail at each node from the very beginning. Given a partial schedule σ, having a span SP_σ and end with a tail of length T_σ, we only need to update the new value of $SP_{\sigma'}$ and $T_{\sigma'}$, when a new batch B_i is appended to σ, that is, $\sigma' = (\sigma, B_i)$. In what follows, we are going to show that the updating of SP_σ and T_σ is much easier than building the Gantt chart from the very beginning.

Suppose that SP_σ and T_σ are the span and tail for a partial schedule σ respectively, and batch B_i is scheduled to follow σ, then the span and tail are updated as in the following equations when the setups of batch B_i have been done on two machines as shown in Figure 4.2.

$$SP^1_{\sigma'} = SP_\sigma + S_{i1} \tag{4.1}$$

$$T^1_{\sigma'} = T_\sigma + S_{i2} - S_{i1} \tag{4.2}$$

In Figure 4.2, $SP_{\sigma'}^2$ and $T_{\sigma'}^2$ are the span and tail when batch B_i have completed its processing on two machines. Again, we have

$$SP_{\sigma'} = SP_{\sigma'}^2 + R_{i1} \tag{4.3}$$

$$T_{\sigma'} = T_{\sigma'}^2 + R_{i2} - R_{i1} \tag{4.4}$$

We denote SP_i^2 the time length of batch B_i spending on machine M_1 with no setup and removal included as shown in Figure 4.2. Then,

$$SP_{\sigma'}^2 = SP_{\sigma'}^1 + SP_i^2 \tag{4.5}$$

Now the remaining problem is how to calculate SP_i^2 and $T_{\sigma'}^2$ when $SP_{\sigma'}^1$ and $T_{\sigma'}^1$ are given. In what follows, we are going to calculate these two values in different cases.

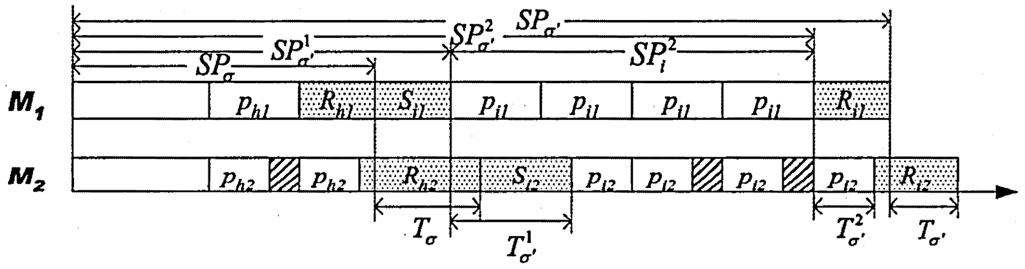


Figure 4.2 Updating of the span SP_σ and tail T_σ

1) $p_{i1} < p_{i2}$

The values of SP_i^2 and $T_{\sigma'}^2$ have different expressions depending on whether batch B_i reaches the steady state or not. We need to compute the values of the steady state batch size in different situations.

(a) $T_{\sigma'}^1 \le p_{i1}$

In this case, as shown in Figure 4.3, the tail $T_{\sigma'}^1$ does not have any contribution to the value of SP_i^2. The value of the steady state batch size b_i^* is the same given by expression (2.15):

$$b_i^* = \lceil cp_{i2}/p_{i2} - p_{i1} \rceil + 1$$

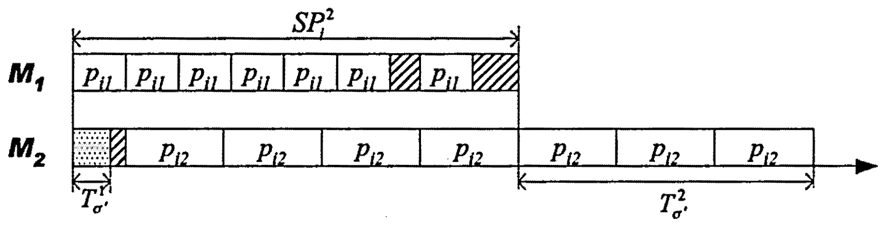


Figure 4.3 The steady state batch size when $T_{\sigma'}^1 \le p_{i1}$ and $p_{i1} < p_{i2}$

(b) $p_{i1} < T_{\sigma'}^1 < (c+1)p_{i1}$

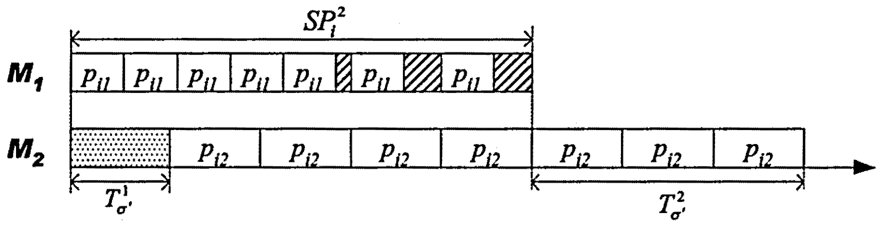


Figure 4.4 The steady state batch size when $p_{i1} < T_{\sigma'}^1 < (c+1)p_{i1}$ and $p_{i1} < p_{i2}$

In this case, as shown in Figure 4.4, the steady state is reached when

$$b_i p_{i1} \le T_{\sigma'}^1 + (b_i - (c+1))p_{i2}$$

Then we have

$$b_i^* = \left|(cp_{i2} + p_{i1} - T_{\sigma'}^1)/p_{i2} - p_{i1}\right| + 1$$

(c) $T_{\sigma'}^1 \geq (c+1)p_{i1}$

In this case, as shown in Figure 4.5, the steady state is reached after the first $(c+1)$ parts, that is,

$$b_i^* = c+1$$

The steady state is always reached in this case since we have the assumption $b_i \geq c+1$.

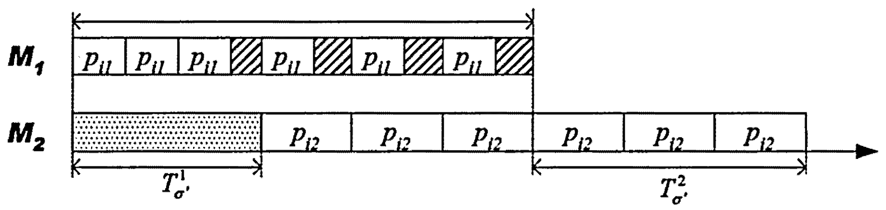


Figure 4.5 The steady state batch size when $T_{\sigma'}^1 \geq (c+1)p_{i1}$ and $p_{i1} < p_{i2}$

The calculations of SP_i^2 and $T_{\sigma'}^2$ are in the following expressions:

If $b_i \geq b_i^*$,

$$SP_i^2 = \max(T_{\sigma'}^1, p_{i1}) + (b_i - (c+1))p_{i2} \tag{4.6}$$

$$T_{\sigma'}^2 = (c+1)p_{i2} \tag{4.7}$$

Else if $b_i < b_i^*$,

$$SP_i^2 = b_i p_{i1} \tag{4.8}$$

$$T_{\sigma'}^2 = \max(T_{\sigma'}^1, p_{i1}) + b_i(p_{i2} - p_{i1}) \tag{4.9}$$

2) $p_{i1} > p_{i2}$

As in the case of $p_{i1} < p_{i2}$, we need to compute the values of the steady state batch size in different situations.

(a) $T^1_{\sigma'} \le p_{i1}$

In this case, as shown in Figure 4.6, the tail $T^1_{\sigma'}$ does not have any contribution to the value of SP_i^2, and batch B_i always reaches the steady state. Then

$$b_i^* = 1$$

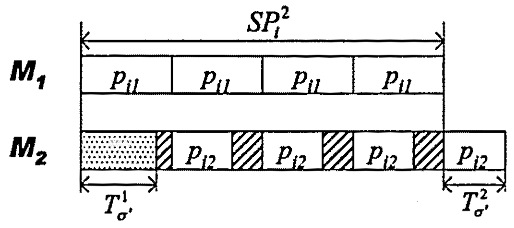


Figure 4.6 The steady state batch size when $T^1_{\sigma'} < p_{i1}$ and $p_{i1} > p_{i2}$

(b) $p_{i1} < T^1_{\sigma'} < (c+1)p_{i1}$

In this case, as shown in Figure 4.7, the steady state is reached when

$$b_i p_{i1} \ge T^1_{\sigma'} + (b_i - 1)p_{i2}$$

Then we have

$$b_i^* = \left| (T^1_{\sigma'} - p_{i1}) / p_{i1} - p_{i2} \right| + 1$$

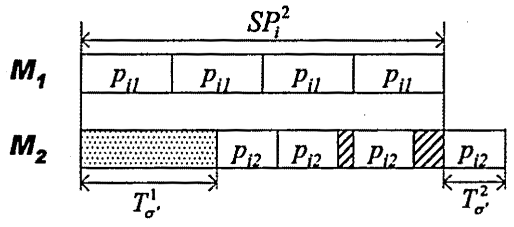


Figure 4.7 The steady state batch size when $p_{i1} < T_{\sigma'}^1 < (c+1)p_{i1}$ and $p_{i1} > p_{i2}$

(c) $T_{\sigma'}^1 \geq (c+1)p_{i1}$

In this case, as shown in Figure 4.8, the steady state is reached when

$$((b_i - (c+1))p_{i1} \geq (b_i - 1)p_{i2}$$

Then we have

$$b_i^* = \lceil cp_{i1} / p_{i1} - p_{i2} \rceil + 1$$

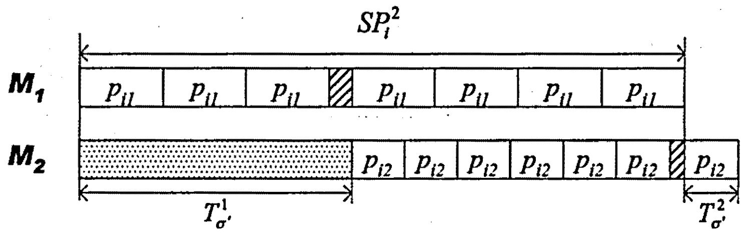


Figure 4.8 The steady state batch size when $T_{\sigma'}^1 \geq (c+1)p_{i1}$ and $p_{i1} > p_{i2}$

For any b_i^*, we have

$$SP_i^2 = \max(T_{\sigma'}^1, (c+1)p_{i1}) + (b_i - (c+1))p_{i1} \tag{4.10}$$

The calculation $T_{\sigma'}^2$ is in the following expressions:

If $b_i \geq b_i^*$,

$$T_{\sigma'}^2 = p_{i2} \tag{4.11}$$

Else if $b_i < b_i^*$,

$$T_{\sigma'}^2 = T_{\sigma'}^1 + b_i p_{i2} - SP_i^2 \tag{4.12}$$

3) $p_{i1} = p_{i2}$

(a) $T_{\sigma'}^1 \le p_{i1}$

In this case, as shown in Figure 4.9, the tail $T_{\sigma'}^1$ does not have any contribution to the value of SP_i^2, and batch B_i always reaches the steady state. Then

$$b_i^* = 1$$

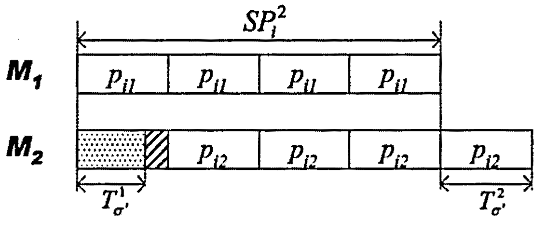


Figure 4.9 The steady state batch size when $T_{\sigma'}^1 \le p_{i1}$ and $p_{i1} = p_{i2}$

(b) $p_{i1} < T_{\sigma'}^1 < (c+1)p_{i1}$

In this case, as shown in Figure 4.10, the steady state will never be reached no matter how large the batch size is, that is,

$$b_i^* = \infty$$

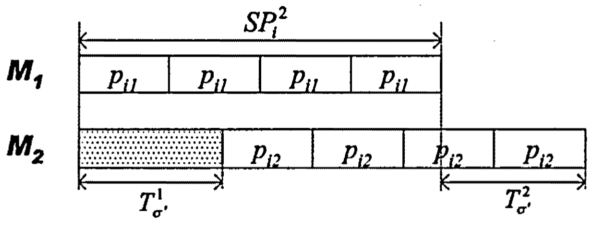


Figure 4.10 The steady state batch size when $p_{i1} < T_{\sigma'}^1 < (c+1)p_{i1}$ and $p_{i1} = p_{i2}$

(c) $T_{\sigma'}^1 \geq (c+1)p_{i1}$

In this case, as shown in Figure 4.5, the steady state is reached after the first $(c+1)$ parts, that is,

$$b_i^* = c+1$$

The steady state is always reached in this case since we have the assumption $b_i \geq c+1$.

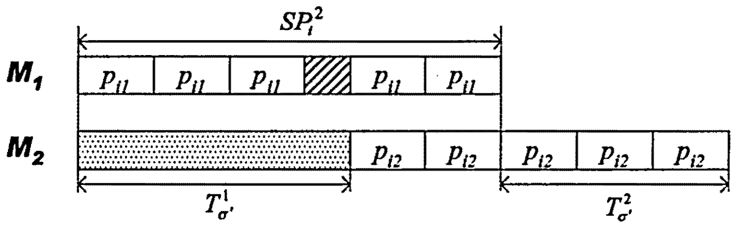


Figure 4.11 The steady state batch size when $T_{\sigma'}^1 \geq (c+1)p_{i1}$ and $p_{i1} = p_{i2}$

The calculations of SP_i^2 and $T_{\sigma'}^2$ are in the following expressions:

$$SP_i^2 = \max(T_{\sigma'}^1, (c+1)p_{i1}) + (b_i - (c+1))p_{i1} \tag{4.13}$$

$$T_{\sigma'}^2 = \max(T_{\sigma'}^1, p_{i1}) + b_i p_{i2} - SP_i^2 \tag{4.14}$$

4.3 Computation of $SP^*(U)$

At node (σ, U), if the value of $SP^*(U)$ can be calculated exactly, there is no need to continue branching from this node. Though not all batches in the set *N* satisfy the steady state condition (2.18), batches in the subset *U* may meet this condition. In this case, we still can apply the Gilmore & Gomory algorithm discussed in the Chapter 2 to obtain the optimal sequencing of *U*. But there is one thing different from the previous situation. As shown in Figure 4.1, the machine M_2 may not be "cleaned up" at the time the first batch in the subset *U* enters the system. In order to consider the tail left from the scheduled batches, we define a dummy batch B_{n+1}, consisting of $b_{i+1} = c+1$ parts having zero setup and removal times on both machines. The processing times on two machines are defined as following:

If $T_\sigma > 0$,

$$p_{n+1,1} = 0 \tag{4.15}$$

$$p_{n+1,2} = T_\sigma / (c+1) \tag{4.16}$$

Else if $T_\sigma < 0$,

$$p_{n+1,1} = -T_\sigma / (c+1) \tag{4.17}$$

$$p_{n+1,2} = 0 \tag{4.18}$$

According to expression (2.6) and (2.7), in either case, the fixed cost SP_{n+1}^{0} is zero. And according to expression (2.8) and (2.9), the coupling cost values are:

If $T_\sigma > 0$,

$$E_{n+1} = 0 \tag{4.19}$$

$$F_{n+1} = T_\sigma \tag{4.20}$$

Else if $T_\sigma < 0$,

$$E_{n+1} = -T_\sigma \tag{4.21}$$

$$F_{n+1} = 0 \tag{4.22}$$

Then from expression (2.11), the switching cost between the dummy batch and the following batch B_i is

$$SP(n+1,i) = \max\{F_{n+1}, E_i\} \tag{4.23}$$

which is exactly the idle time of the first scheduled batch of *U*. The switching cost between the preceding batch B_h and the dummy batch is

$$SP(h,n+1) = \max\{F_h, E_{n+1}\} \tag{4.24}$$

which is the tail of the batch schedule.

After applying the Gilmore & Gomory algorithm, the batch scheduled first in *U* is the one following the dummy batch in the cyclic path, and the batch scheduled last is the one preceding the dummy batch in the cyclic path.

4.4 Lower Bounds of $SP^*(U)$

At node (σ, U), if not all batches in the subset of U can satisfy the steady state condition (2.18), the Gilmore & Gomory algorithm as stated in section 4.3 does not guarantee an optimal solution. In this case, we need to find a lower bound of $SP^*(U)$. If the lower bound of $SP^*(U)$ plus SP_σ exceeds the best-known upper bound, node (σ, U) cannot be optimal, and hence can be eliminated. In Chapter 3, we have already described two approaches in finding lower bounds of $SP^*(U)$. However, as in section 4.3, one thing different from the previous situation is that machine M_2 is still being occupied by a tail of the scheduled sequence σ when the first batch of U enters the system. In what follows, we are going to adjust the two algorithms to this situation.

1) Time Reduction Lower Bound

If some batch $B_i \in U$ cannot satisfy the steady state condition (2.15), we reduce the smallest processing time of p_{i1} and p_{i2} until

$$b_i \geq \left\lceil \frac{c \max(p_{i1}, p_{i2})}{|p_{i1} - p_{i2}|} \right\rceil + 1$$

The tail of the scheduled sequence σ can be substituted by a dummy batch that has been defined in section 4.3. After these alterations, we apply the Gilmore & Gomory algorithm to obtain a sequence of U. The makespan of this sequence is a lower bound of $SP^*(U)$.

2) Infinite Buffer Lower Bound

Another way to find a lower bound of $SP^*(U)$ is to relax the buffer limitation condition. If we suppose that the intermediate buffer has unlimited capacity, the Infinite Buffer algorithm described in Chapter 3 optimally sequences the unscheduled subset U. We denote this sequence σ'. When the tail T_σ of the scheduled sequence σ is considered, two different situations arise.

(a) $T_\sigma \geq 0$

In this case, the tail is defined as a dummy job B_{n+1} with processing times

$$p_{n+1,1} = 0 \tag{4.25}$$

$$p_{n+1,2} = T_\sigma \tag{4.26}$$

We claim that the sequence (B_{n+1}, σ') is optimal for the batch set $\{B_{n+1}\} \cup U$ since, according to Johnson's rule, a job with processing time zero on the first machine is always scheduled first in the sequence. Therefore, in this case, we do not need to make any change on our Infinite Buffer algorithm in solving the unscheduled subset U.

(b) $T_\sigma < 0$

In this case, the tail of the scheduled sequence σ is the same as a dummy job B_{n+1} with processing times

$$p_{n+1,1} = -T_\sigma \tag{4.27}$$

$$p_{n+1,2} = 0 \tag{4.28}$$

However, in this case, the Infinite Buffer algorithm may be not optimal since the dummy job, which is supposed to be scheduled last in the sequence according to the Johnson's rule, is now in the first position of the sequence. We cannot move the dummy job to the last position of the schedule since, in reality, the dummy job is the tail of the scheduled sequence σ, whose first position is fixed. In this case, we need to make some alteration on our Infinite Buffer algorithm.

For the unscheduled batch set *U*, after applying the Infinite Buffer algorithm, we obtain a sequence σ' and a makespan $SP_{\sigma'}$. This sequence is optimal when no previous tail and no buffer limitation are assumed. In the sequence σ', there is no idle time between batches on the machine M_1. If we schedule jobs on the second machine M_2 as late as possible while keeping the position of last scheduled job unchanged, the resulting schedule is still optimal. After that, four possible resulting schedules are shown in Figure 4.12. In both figure (a) and figure (b), there are idle times on the second machine M_2 before the setup of the first batch in *U*. In this situation, the negative tail T_σ cannot help to shorten the makespan $SP_{\sigma'}$. Hence the Infinite Buffer algorithm is optimal in both (a) and (b). In figure (c), there is no idle time on the second machine, but the ending tail of the whole schedule $T_{\sigma'}$ is negative. In this case, though the negative tail of the scheduled sequence σ may help to make the completion time of all batches on machine M_2 earlier, it is not able to shorten the makespan $SP_{\sigma'}$, which is determined by the busy time on machine M_1 in this case. Hence the Infinite Buffer algorithm is also optimal in (c).

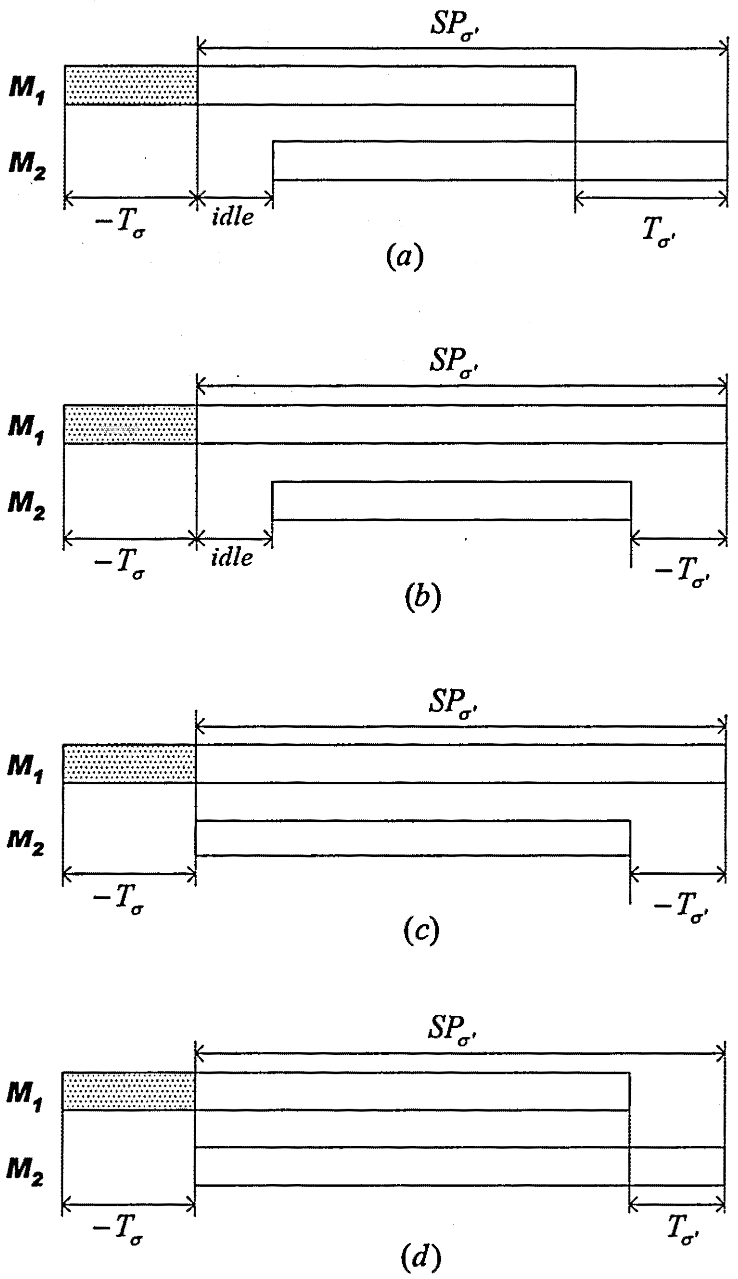


Figure 4.12 Four possible resulting schedules of the Infinite Buffer algorithm

However, in figure (d), by occupying the negative tail T_σ, earlier scheduling of the first batch in *U* on machine M_2 may cause a shorter makespan. In this situation, the Infinite Buffer algorithm does not guarantee an optimal solution. Some alterations are needed to determine the optimal sequence σ' of the subset *U* so that the negative tail T_σ is maximally "utilized".

Suppose batch B_i is assigned to be in the first position following the scheduled sequence σ. Again, we schedule jobs of B_i on the second machine M_2 as late as possible as shown in Figure 3.2. Suppose that $\alpha_i > 0$, as we discussed before, there is an idle time on the second machine, and the negative tail T_σ has not been utilized. In other words, only batches with $\alpha_i = 0$ can possibly reduce the idle time left by the previous scheduled batch on the second machine. According to the Johnson's rule, batches with $\alpha_i = 0$ are assigned to be in the first positions of the whole schedule. If there are more than one batches with $\alpha_i = 0$, we need to rearrange the positions among them so that the total idle time is minimized.

For a batch B_i with $\alpha_i = 0$, the maximal time length that could possibly fill the negative tail left by the previous scheduled batch is constrained by the fact that a job can start its processing on machine M_2 only after it finishes its processing on the first machine. As shown in Figure 4.13, we denote that maximal time length as d_i. Then if $p_{i1} < p_{i2}$,

$$d_i = S_{i2} - (S_{i1} + p_{i1}) \tag{4.29}$$

Else if $p_{i1} \geq p_{i2}$,

$$d_i = S_{i2} + (b_i - 1)p_{i2} - (S_{i1} + b_i p_{i1}) \tag{4.30}$$

Now an optimal sequencing for the subset U is obtained by scheduling batches with $\alpha_i = 0$ in non-increasing order of their d_i's, and keeping all other batches in the same order as the result of the Infinite Buffer algorithm.

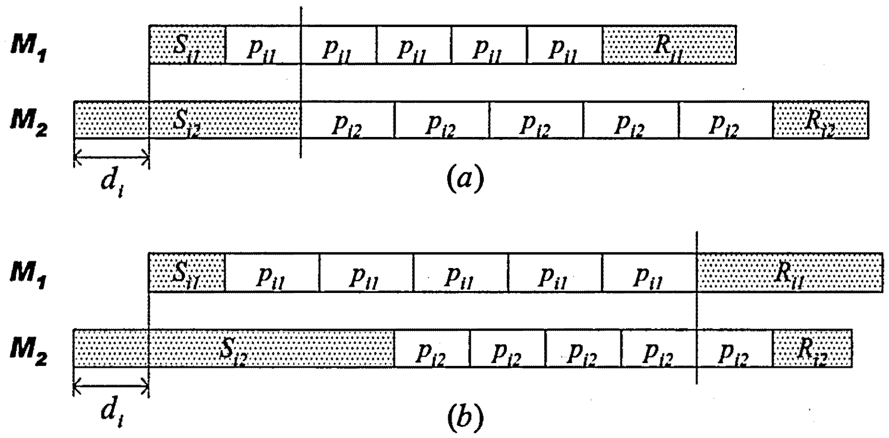


Figure 4.13 The maximal time length d_i

4.5 A Branch and Bound Algorithm

```
begin
    obtain a sequence $\sigma$ by applying Gilmore & Gomory algorithm;
    upper_bound := $C(\sigma)$;
    while not all nodes have been searched do
    begin
        determine the next node $(\sigma, U)$ to be valued;
```

```
            if all batches in U reach the steady state then
              begin
                    find $SP^*(U)$ by applying Gilmore & Gomory algorithm;
                    if $SP_\sigma + SP^*(U) < upper\_bound$ then
                          $upper\_bound := SP_\sigma + SP^*(U)$;
                    delete this node;
              end;
            else
              begin
                    find $SP^1(U)$ by applying Time Reduction algorithm;
                    find $SP^2(U)$ by applying Infinite Buffer algorithm;
                    if $SP_\sigma + max(SP^1(U), SP^1(U)) > upper\_bound$ then
                          delete this node;
                    else
                          find $SP^3(U)$ by applying Gilmore & Gomory algorithm;
                          if $SP_\sigma + SP^3(U) < upper\_bound$ then
                                $upper\_bound := SP_\sigma + SP^*(U)$;
              end;
        end;
end;
```

4.6 An Example

In this section, we are going to present a numerical example to illustrate the procedure of the branch and bound algorithm described in the previous sections. The example data is described in Table 4.1. In this example, we have six batches to be processed, and the buffer size is three. The enumeration tree is shown in Figure 4.14. At the root node, we apply the Gilmore and Gomory algorithm to obtain a feasible sequence $B_1, B_4, B_3, B_5, B_6, B_2$, and a corresponding upper bound of 1683. After applying of the Time Reduction algorithm and Infinite Buffer algorithm, we obtain two makespans, and the maximum of them is our starting lower bound. In Figure 4.14, each node is associated with a partial schedule and a lower bound. The optimal solution is obtained at the sequence $B_1, B_3, B_5, B_6, B_4, B_2$, with a corresponding makespan of 1677.

B_i	1	2	3	4	5	6
b_i	14	13	6	5	8	16
p_{i1}	9	16	20	18	24	35
p_{i2}	11	16	16	21	43	28
S_{i1}	23	32	73	23	5	21
S_{i2}	34	32	56	21	8	22
R_{i1}	43	73	12	14	13	7
R_{i2}	41	14	23	27	12	9
c	3					

Table 4.1 Data of the example

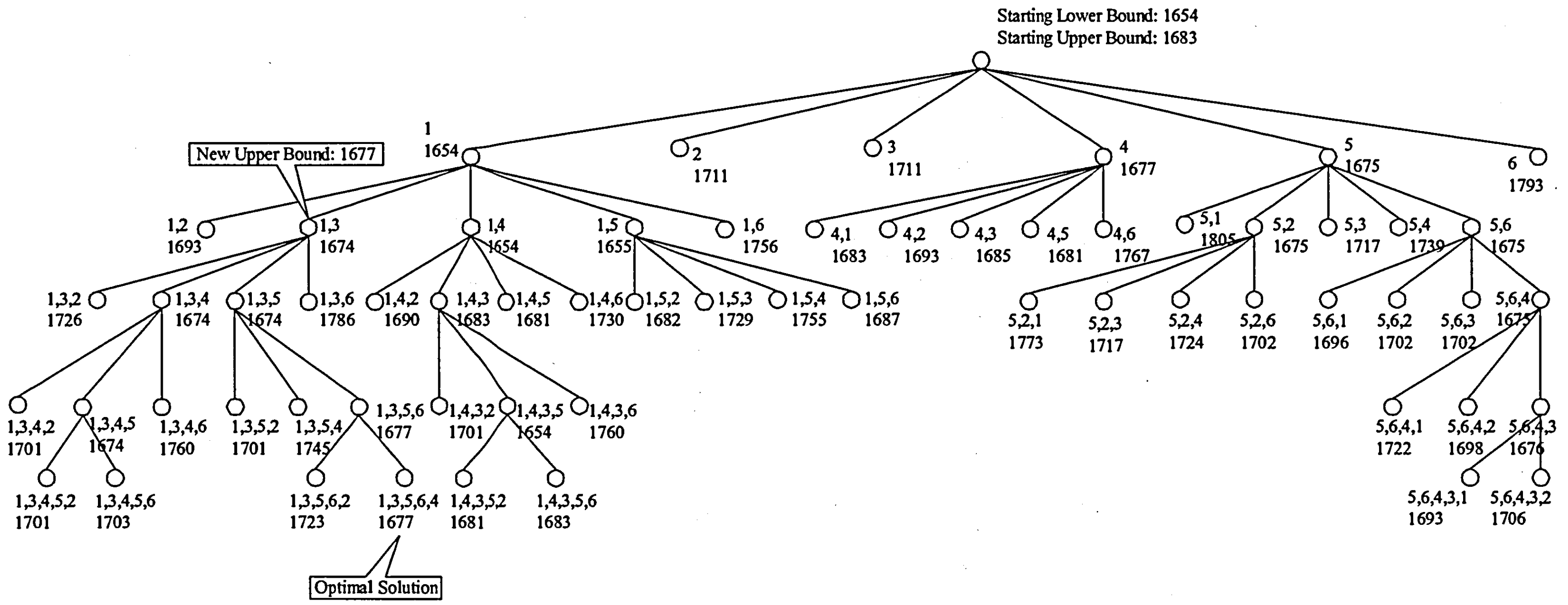


Figure 4.14 The enumeration tree of the example

Chapter 5

Numerical Experiments

5.1 Outline

In this chapter, we report the results of a series of numerical experiments performed to analyze the effectiveness of the branch and bound algorithm developed in the previous chapters. There are several parameters characterizing the batch scheduling problem: the number of batches, intermediate buffer capacity, batch sizes, and the values of processing times. As we can see in the following section, the time needed to find an optimal solution is the most sensitive to number of batches. The intermediate buffer capacity affects both the running time and the optimal value. The batch sizes and the values of processing times affect the performance of the program by their contributions to the percentage of batches that reach the steady state. Whether a batch reaches the steady state or not is decided by the buffer capacity as well. Therefore, in our experiments, we need to examine the performance of the program by: (a) the number of batches; (b) buffer capacity; (c) the percentage of batches that reach the steady state.

In this chapter, we also compare the branch and bound algorithm with some other heuristic approaches to show how much improvement our method gains in various cases. Our branch and bound algorithm uses the depth-first search strategy in searching all nodes of an enumeration tree. In this chapter, we also develop another algorithm, which applies best-first search strategy to branch the enumeration tree. We present a performance comparison between these two search approaches.

In the following experiments, processing times are random numbers between 10 and 150, setup and removal times are random numbers between 50 and 600, and batch sizes are random numbers between 8 and 200. All random numbers are generated by computer in pseudo-uniform distribution. All experiments are performed on a Dell Dimension 4550 personal computer with Pentium 4 processor at 2.53GHZ speed and 640 M RAM. Better performance is expected if a higher-speed computer is available.

5.2 Numerical Experiments

1) The effect of the number of batches

In this experiment, we fix the buffer size at five. We divide the experiment in five categories: the percentages of batches that reach the steady state are 10%, 30%, 50%, 70% and 90% respectively. In each category, we begin with the problem of one batch, and increase one more batch at each step. We select five groups of data that satisfy the percentage of that category at each step, and get an average running time. For a problem with fourteen batches or less, the running time is below one minute for any input data.

But when the number of batches is larger than fourteen, the running time grows exponentially as shown in Figure 5.1.

2) The effect of buffer capacity

In this experiment, the number of batches and processing times are fixed. We begin with the problem of zero-buffer, and increase capacity by one buffer position at each step. In the case of zero-buffer, the problem is equivalent to the two-machine blocking flowshop, which can be solved in $O(n \log n)$ time. The running time grows as the buffer size increases, as shown in Figure 5.2. This is because, from equation (2.18), batches are likely not to reach the steady state condition as the buffer size increases. The running time drops fast after the buffer capacity reaches a certain number, since from that point on, the buffer size is large enough so that blocking on the first machine is not likely to happen. The Johnson's algorithm provides an optimal solution in that case. The value of makespan decreases as the buffer size enlarges, and after a certain point (thirteen in this case), the value keeps constant since no blocking occurs thereafter, as shown in Figure 5.3.

3) The effect of the percentage of steady state

In this experiment, we fix the buffer size at five and the number of batches at eighteen. We run the program at various processing times and setup and removal times, and calculate the running time and the percentage of batches that reach the steady state at each run. When all batches reach the steady state condition, the branch and bound algorithm terminates right at the root node. The running time grows as the number of batches that can reach steady state condition decreases. Around the point of forty percent, the running time drops because from that point on, Johnson's algorithm provides good upper bound. As shown in Figure 5.4, for batches at the two ends of the diagram, the branch and bound algorithm can find the optimal solution quickly.

4) Comparison with other heuristic approaches

This experiment is designed to show how much improvement that our method gains over the other heuristic approaches. Again, we fix the number of batches at 18 and the buffer size at 5. We run the program at various processing times and setup and removal times, and calculate the makespans and the percentage of batches that reach the steady state at each run. As shown in Figure 5.5, for batches with low percentage of steady state, the Johnson's algorithm is close to the optimal solution, while the steady state algorithm has as much as thirty-two percent higher than the optimal value. For batches of high percentage of steady state, the steady state algorithm tends to be optimal, while the Johnson's algorithm has as much as thirty-six percent higher than the optimal value. On average, both heuristic algorithms have about fifteen percent higher values in makespan than the optimal solution. Based on this information, we can see that heuristic approaches can be "good" only in some particular situations.

5.3 Best-First Search

We apply depth-first strategy to search the enumeration tree. Depth-first performs a deep probe, creating a path as long as possible, and backs up one node to initiate a new probe when it can mark no new node from the tip of the path. On the contrary, breadth-first strategy expands all possible branches from a node in the search tree before going deeper into the tree. Besides these two commonly used search strategies, another one is best-first search, which always select the node that seems most promising. In our problem, at each step of the branch and bound algorithm, we select the node corresponding to the partial schedule for which the lowest value of lower bound has been found. When several nodes have the same lower bound, the deepest among them was chosen. We develop another branch and bound algorithm based on best-first search strategy. The mechanism of the strategy is shown in the following example.

Example

In this example, we still apply the example data Table 4.1 in Chapter 4. As shown in Figure 5.6, the procedure of the best-first search is developed in the following way. At the root node, a lower bound of 1654 and an upper of 1683 has been found. The first node to be searched is node 1, which also has a lower bound of 1654. Since node 1 is deeper than the root node in the enumeration tree, we search the nodes derived from node 1. Node 1-2 has a lower bound of 1693, which is larger than the upper bound of 1683. Then node 1-2 is eliminated. At node 1-2, we find a new upper bound of 1677, therefore, the upper bound is updated at this value. Since the lower bound (1674) of node 1-2 is larger than current lowest lower bound (1654), we continue to search other nodes derived from node 1. At node 1-4, we also find a lower bound of 1654. Since node 1-4 is deeper than node 1 in the enumeration tree, the next nodes to be searched are derived from node 1-4. The same procedure is developed until we found the optimal solution at node 1-3-5-6-4.

In this example, total 57 nodes have to be searched to find the optimal solution, compared with the depth-first search algorithm, which only searches 18 nodes.

A comparison of two search algorithms is shown in Figure 5.7. We run the best-first search algorithm under same conditions as in example 1. For batch scheduling problems with small batch number, the depth-first search has better performance than the best-first search. The reason is that the best-first search spends much more time in storing the enumeration tree structure, which trades off the gain of "wise" search policy. For problems with batch number of twenty, the best-first search runs faster, about seventeen percent improvement over the depth-first search on average. However, for problems with twenty-one batches, the depth-first algorithm may take a couple of days to obtain an optimal solution, but the best-first search runs out of memory on the computer.

Generally, the best-first search can have a better performance than the depth-first search for a problem with large batch number. However, since it needs to memorize the structure of the enumeration tree, more memory is required for this search strategy. It may become impractical for a large-scale problem. On the contrary, the depth-first search has the following advantages: (1) we can always get a feasible solution, even when the number of nodes are to be searched is extraordinarily large; (2) For practical problems, the optimal solution usually occurs deep in the tree; (3) The search scheme is easy to be applied, since we do not need to memorize the structure of the tree.

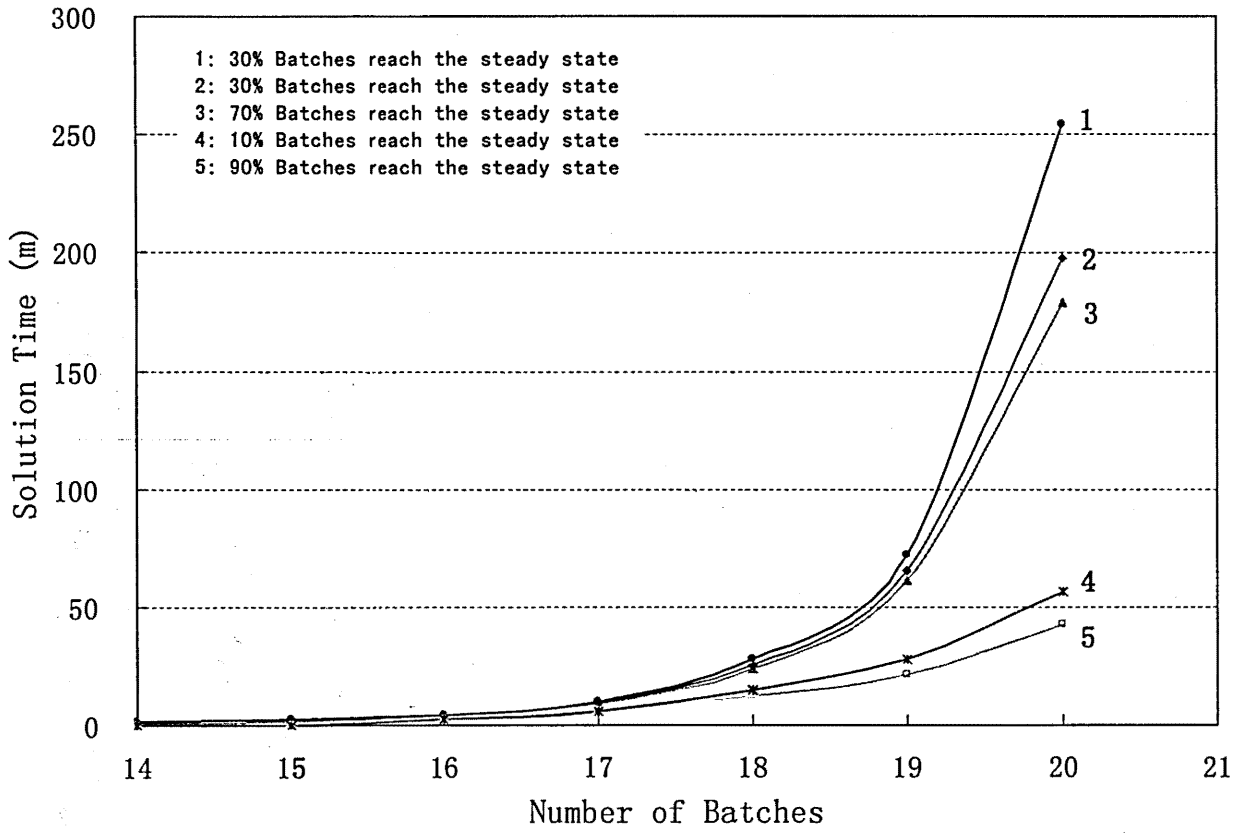


Figure 5.1 Solution time corresponding to batch number of batches

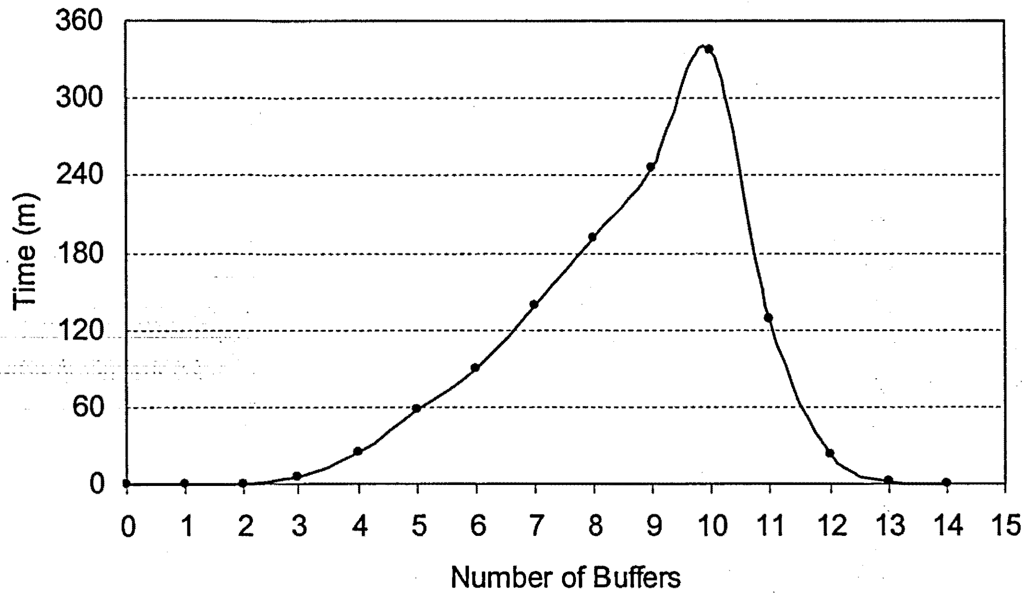


Figure 5.2 Solution time corresponding buffer capacity

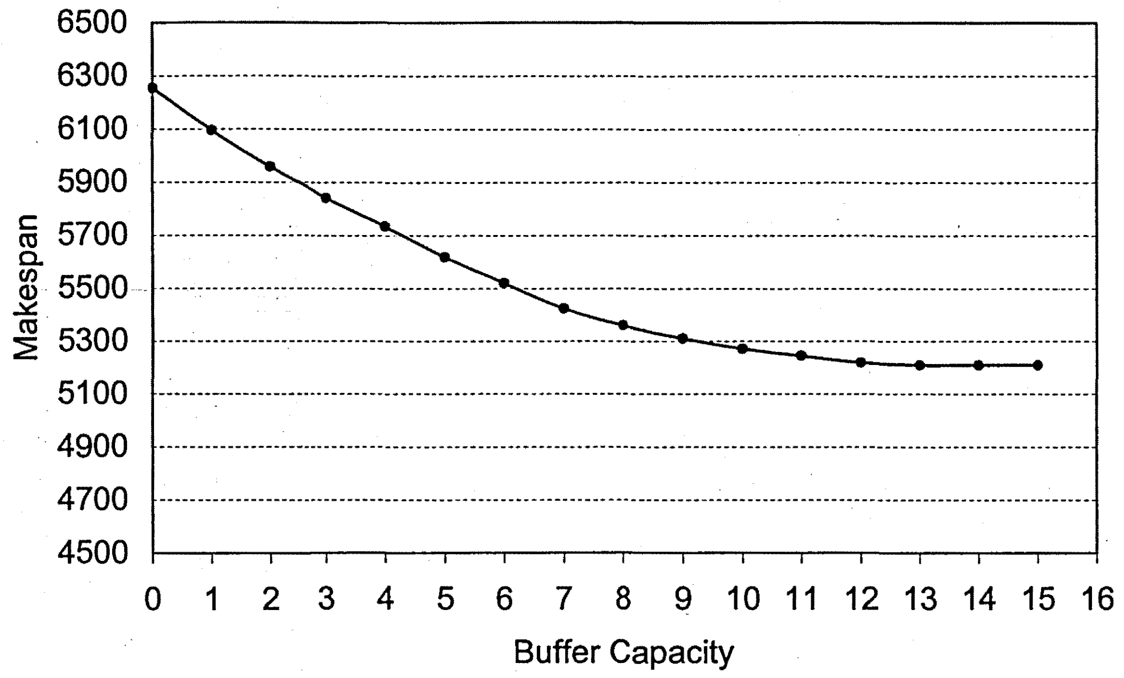


Figure 5.3 Makespan corresponding to buffer capacity

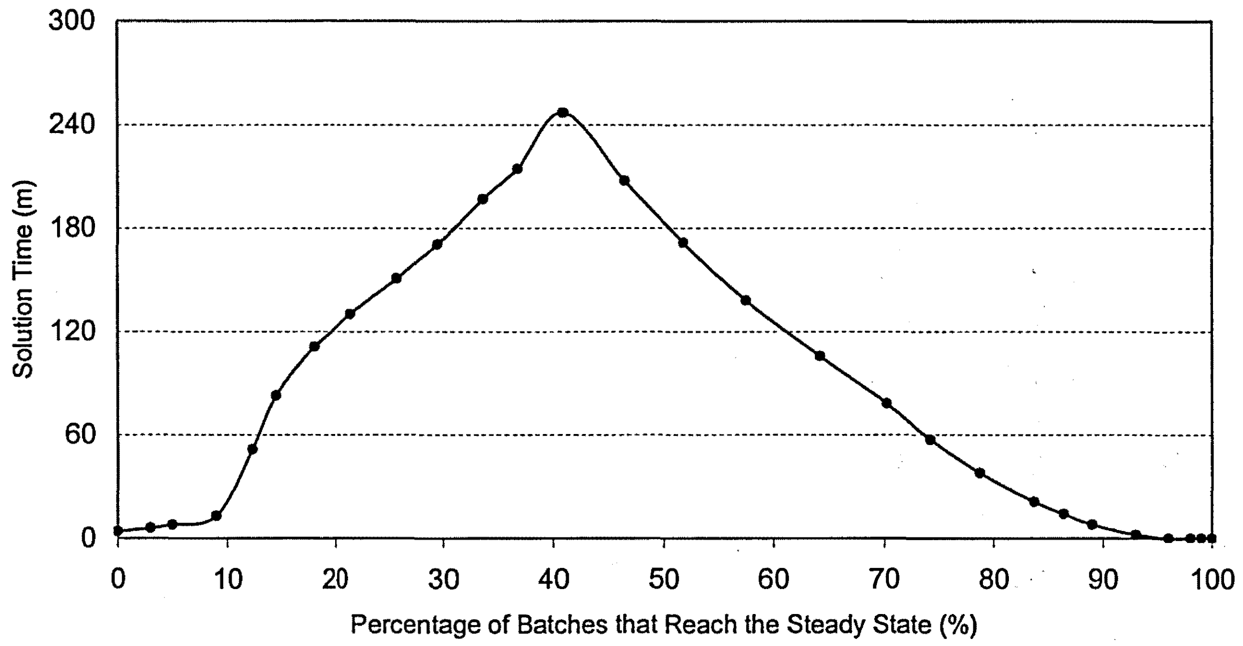


Figure 5.4 Solution time corresponding to percentage of batches that reach the steady state

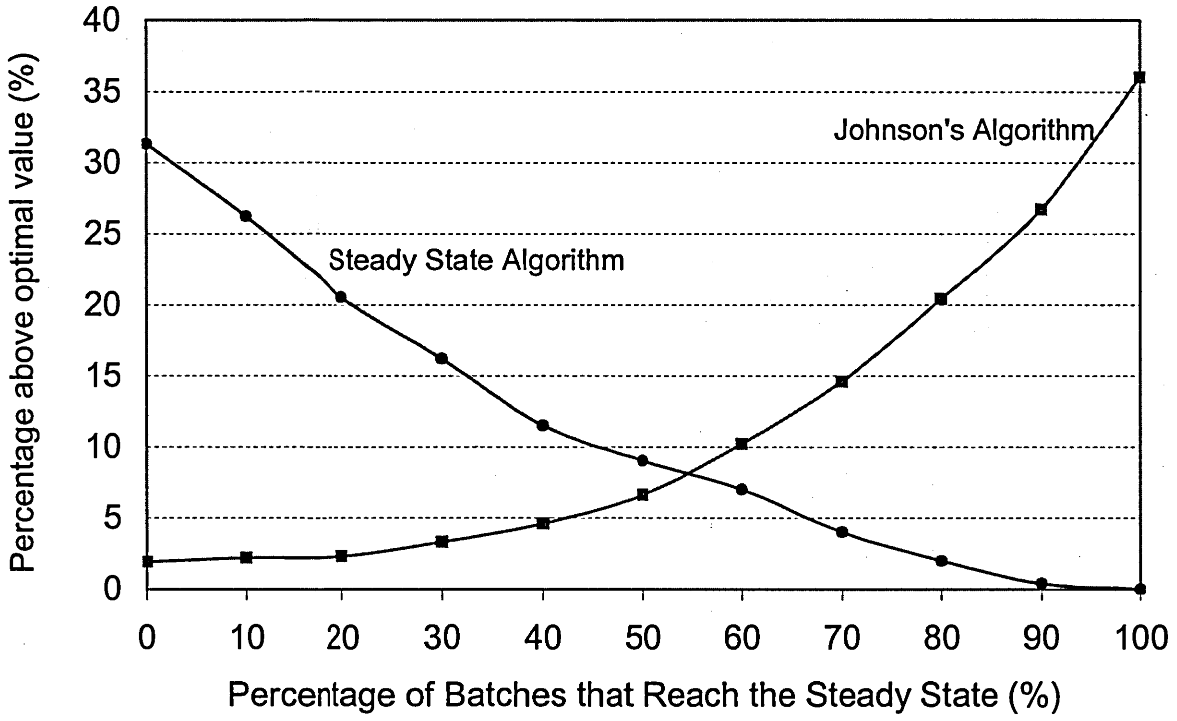


Figure 5.5 Makespans of the heuristic algorithms

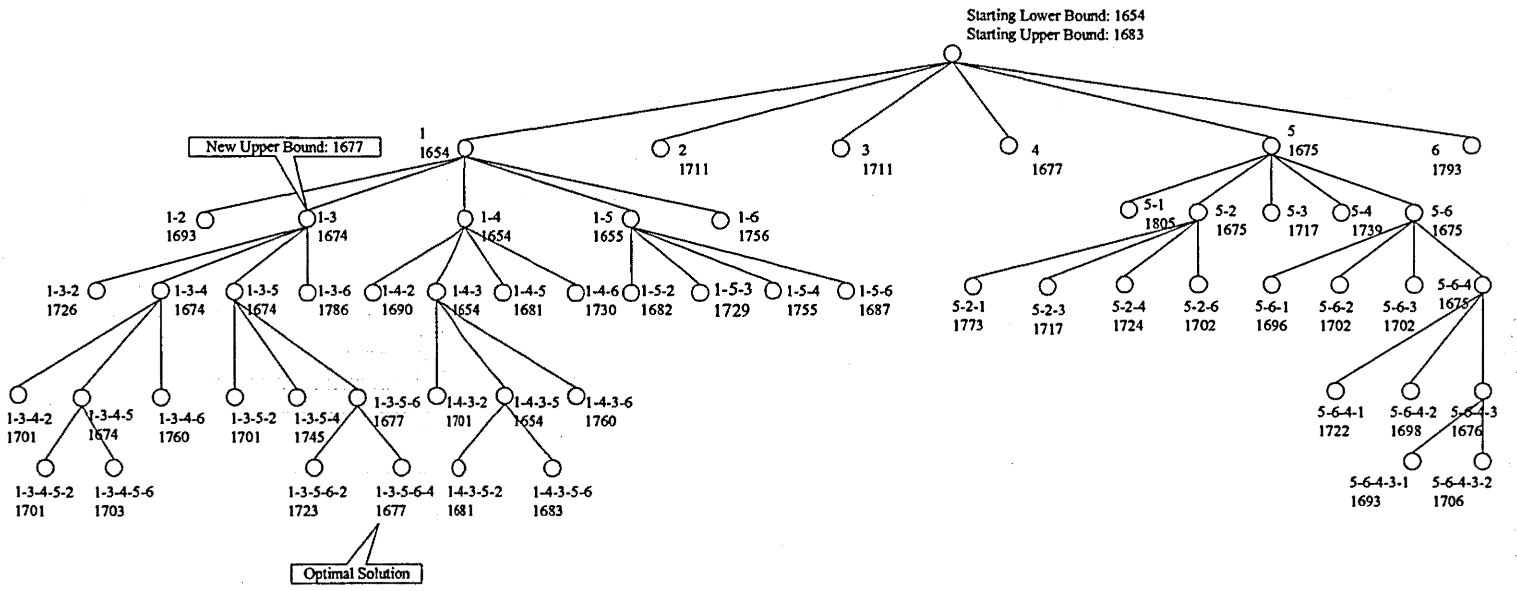


Figure 5.6 The enumeration tree of the best-first search

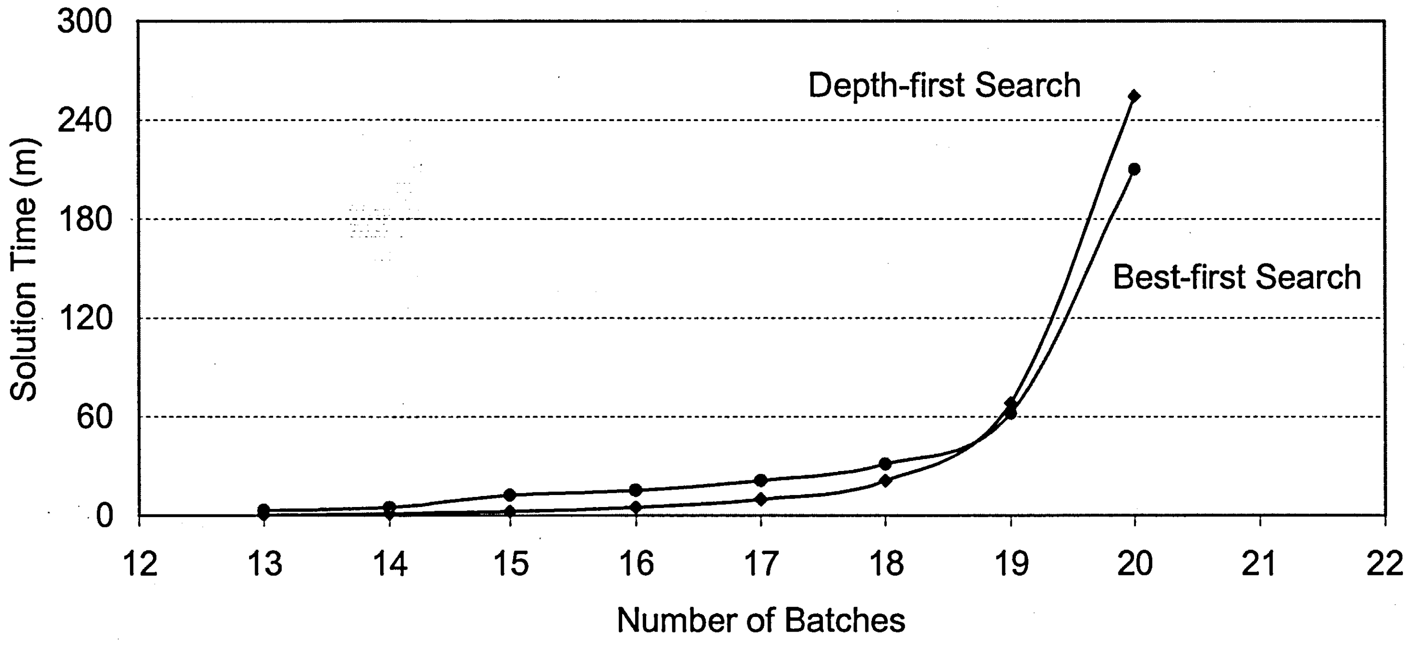


Figure 5.7 Computation Requirements for Algorithms: Depth-first vs. Best-first

Chapter 6

A Case Study

6.1 Background

PCB assembly is a complex task involving the placement of up to hundreds of electronic components in different shapes and sizes at specific locations on the board. The process of PCB assembly can be carried out in an automated flow line with several stages, which include printing, placement, heating and testing, as illustrated in Figure 6.1. Since placement machines are usually much more expensive than other equipment in the assembly line, the component placement is often the bottleneck of the production line, and consequently it becomes the focus of efforts to improve production flow.

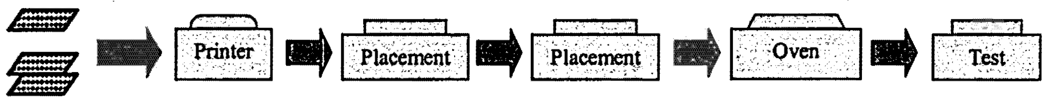


Figure 6.1 A PCB assembly line

Although placement machines have various configurations, most of them consist of three components: a table on which the PCB is attached, a feeder carriage that holds components, and a head that picks components from the feeder and places them on the PCB. There are four operational decisions to be made during the process of placement. *Machine assignment* decides which components to be placed on which machine. *Feeder assignment* settles which components to be installed in which feeder for each machine. *Placement sequencing* decides the order of components to be placed on a given board for each machine. *Board sequencing* determines the processing order of different types of boards on the production line. In this dissertation, our research focuses on the board sequencing problem, that is, we assume that all other operational decisions have been given, and we try to optimize the board sequence on machines. A comprehensive review of the literature in PCB assembly optimization problems can be found in [McGinnis et al., 1992], [Ammons et al., 1997], [Leon and Peters, 1998], [Ahmadi and Kouvelis, 1999] and [Ellis et al., 2001].

Usually there is more than one placement machine in a PCB assembly line. A popular line configuration consists of two different types of placement machines: one type of machine places commonly used components, such as resistors and capacitors, at high speed; the other type of machine is more flexible but at a lower speed. Large chips and other irregular components are usually placed on this type of machine. This research considers an assembly line with two placement machines. The two machines are connected by a synchronized conveyor. The placement machines and conveyor form a two-machine flowshop with limited capacity intermediate buffer, as shown in Figure 6.2.

The capacity of the buffer is the same as the number of board positions on the synchronized conveyor.

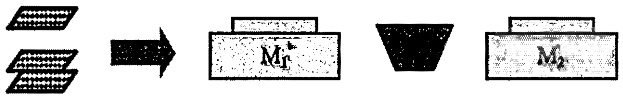


Figure 6.2 Two-machine flow shop with limited buffer

In PCB assembly, boards are usually grouped in batches. We assume that boards in the same batch are all identical. Since setup times between boards in a same batch can be regarded as a part of processing times, for simplicity, we assume that no setup is required between boards in the same batch. However, we assume that there is a changeover between two different batches. This changeover includes the activities associated with the removal and installment of components in feeder slots and machine adjustment for a given board type. The changeover time is often large relative to the placement time of a single board. And furthermore, the changeover time is greatly influenced by the PCB production sequence. Ammons et al. [1997] categorize the strategies for setup management as follows:

1. *Single setup* in which a group of machines is configured to produce a family of PCBs using a single setup. There are two possible single setup strategies:
   a) *Unique setup* in which the family contains only one product type.
   b) *Family setup* in which the family comprises several product types.
2. *Multiple setup* in which because a limited component staging capacity on the placement machines prohibits applying the single setup strategy, some additional

setups must be performed within a family. There are two possible multiple setup strategies:

a) *Decompose and sequence* in which the family is divided into sub-families which are then sequenced to minimize the incremental setups between the subsets.

b) *Partition and repeat* in which the required components are partitioned into subsets restricted by machine capacity.

In our case study, we assume a unique setup strategy. The advantage of the unique setup strategy is that the placement time can be minimized for each board. Also the makespan is not affected by the board sequence when using unique setups [Leon and Peters, 1998]. The changeover times can be large since all components from the previous product are removed before starting the setup of a new product. However, this can be compensated by *offline setups*. An operator can set up the required components for a product in an offline feed bank while the machine is building a different product. The operator then trades the offline feeder bank for the online feeder bank, and can start building the new product immediately [Palm, 1996]. An offline setup usually takes a couple of minutes to load the feeder bank into the machine.

Another setup strategy to improve the machine utilization is *sequential changeover*. In a sequential changeover, a placement machine starts to change over to a new batch as soon as it finishes the last board of the current batch, which means two different PCB types may be on the assembly line at the same time [Rowland, 2003]. We apply sequential changeover strategy in this research.

6.2 Case Study

National Instruments (NI) assembles printed circuit boards for their customers. On an average, NI produces 10 types of boards each day. The number of boards for each type varies from 5 to hundreds, with an average of 60. These numbers feature a low-volume high-mix manufacturing. The boards are built to stock following a min-max policy. A push system is applied for most non-urgent batches. The operations people schedule 2-day release of boards and WIP each day to push the products towards the finished goods stock. NI would like to know what, if any, impact different scheduling strategies may have on the production flow.

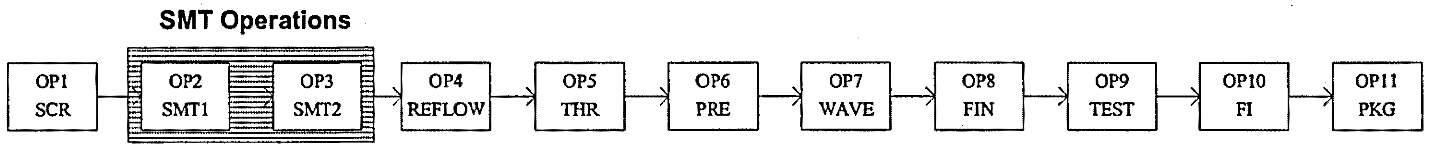
NI has four PCB assembly lines in operation. Three of these are surface mount lines that contain identical machine configuration. Another line has a through-hole machine right after SMT machines. Each production line has two different SMT placement machines. The first placement machine is a chip-shooter, and most passive components are placed on this machine. The second one is a flexible machine, and ICs are placed on this machine. The operation flow for each PCB through an assembly line is illustrated in Figure 6.3. Two SMT machines are the bottlenecks of each production line, and hence they are the focus of efforts to be optimized.

NI applies unique setup strategy, that is, a single setup is performed for one type of product. All setups are performed off-line. On estimation, it takes 30 seconds to load/unload a feeder into/from an SMT machine. Rolling changeover is also applied. A machine begins to perform a setup for the next batch as soon as the last job of the current batch leaves the machine.

In this experiment, we schedule twenty batches for each production line which take about 24 hours in continuous production time. We calculate the makespans by our branch and bound algorithm, and compare them with NI's scheduling. The experimental data and results are in Appendix C. Table 6.1 shows that our approach provides approximately 6-7% improvement over current schedules.

	Line 1	**Line2**	**Line3**	**Line4**
NI Scheduling (h)	26.4	23.7	29.8	21.3
Optimal Scheduling (h)	24.7	22.3	28.1	19.8
Percentage of Improvement (%)	6.5	6.1	5.7	7.1

Table 6.1 Experiment results



Operation	Details
OP1	Solder screening
OP2	Placement of passive components
OP3	Placement of ICs
OP4	Reflow oven
OP5	Insertion of thru-hole components (optional)
OP6	Hand placement of components
OP7	Wave solder
OP8	Final Assembly
OP9	Tests
OP10	Final Inspection
OP11	Board packaging

Figure 6.3 Operations for each PCB assembly line

6.3 Discussion

In this case, the setup times and removal times are all identical. For a more general case, suppose that both setup and removal times are identical for each batch (but can be different between batches), as shown in Figure 6.4, $R_1 = R_2$, $S_1 = S_2$, then we have $T_1 = T_2$, which implies the setup and removal times do not have any impact on the optimality of the overall schedule. Therefore, if we do not consider the setup and removal times for all batches, the optimal sequence of all batches is the same as when setup and removal times considered.

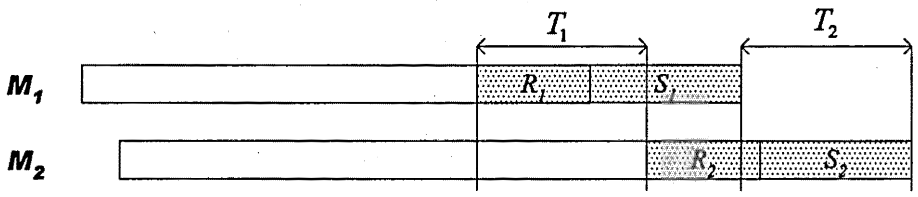


Figure 6.4 Identical setup and removal times

Chapter 7

Conclusions

7.1 Summary

In this dissertation, we study the flowshop batch scheduling problem for the two machine case. Based on previous research, we develop a steady state optimization algorithm, which takes both sequence-independent setup and removal times into consideration. We also develop two heuristic algorithms that provide lower bounds of the optimal solution. A branch and bound algorithm is designed to find the optimal solution for any general case. Theoretically, the branch and bound algorithm can always obtain an optimal solution for any general cases (on our personal computer, we can solve a 20-batch problem within several hours). Even for instances with large numbers of batches, our algorithm can always obtain feasible solutions, and most of time, rather good solutions. Finally, we perform a series of numerical experiments to show the effectiveness of our approach. Compared with the depth-first search algorithm, a best-first search algorithm is also developed, and a performance comparison between these two approaches has been performed.

7.2 Contributions

This research finds an optimal batch sequence in a PCB assembly line such that the total flow time of all jobs is minimized. The main contributions include:

1) We extend the steady state optimization approach of the previous research work. In our model, we include batch setups and removals. Since the tail T_σ can be negative in our case, we solved the problem in a higher complexity.
2) We develop an optimal algorithm for the batch scheduling problem of a two-machine flowshop with unlimited buffer. We also extend the algorithm for the situation in which one of the two machines is not available at time zero (this is the same situation with a tail T_σ left from the partial schedule σ as described in Chapter 4). This algorithm provides a good lower bound when most of batches to be scheduled have close processing times on two machines. We also develop the time-reduction algorithm, which provides good lower bound when most of batches to be scheduled can reach the steady state.
3) Our algorithm is designed for one type of PCB assembly scheduling problems. More specifically, the problem has the following features:
   a. The flowshop has two SMT machines, possibly non-identical ;
   b. Unique setup policy is applied;

c. Rolling changeover strategy is applied;

d. A group of orders is available to be scheduled with no due date specified;

e. The objective is to minimize makespan.

Since the conditions of PCB assembly can vary a great deal, there are many different PCB scheduling problems. The percentage of the PCB assembly systems that meet all these features would be rather small. However, the electronics industry has been growing rapidly since decades ago. Nakahara [1999] indicated that the annual growth rate of worldwide PCB production has exceeded 20% since 1984, to a total value of roughly $35 billion in 1998. The total value saved by improving machine utilization by using our approach can still be high.

7.3 Future Research

There are several points to be explored in the future research.

1) The efficiency of our branch and bound algorithm is based on input data. The running time increases as the number of small size batches (for example, smaller than buffer size *c*) grows, since in that case, the steady state optimization algorithm cannot eliminate nodes on the enumeration tree quickly. Some other methods in finding new lower bounds for small batch size cases should be one of our further studies.

2) Contrary to time-reduction algorithm, another approximation approach is to increase the smaller processing time between p_{i1} and p_{i2} of an amount such that no blocking happens for that batch B_i. When no blocking happens for all batches, the problem can

be solved by the infinite-buffer algorithm we developed. The difficulty of this approach is to find out how much to increment of the processing time for each batch. This approach can be called the time-increment algorithm. The approximation result provides an upper bound of the optimal solution.

3) This research investigates a limited-buffer flowshop with two machines. In reality, some PCB production lines have more than two machines in series. Our model can be applied in any situation where each bottleneck is composed of two machines. If there are three or more machines in series to be considered, our flowshop model needs to be extended. However, since a general case of three-machine flowshop with unlimited buffer is already *NP*-hard, this brings a significant hardness to the problem. Both our steady state optimization approach and heuristic algorithms need to be further explored.

4) In reality, all machines are subject to random breakdowns. How the stochastic nature of the system affects the performance of our approaches should be another research topic.

Appendix A: Gilmore and Gomory Algorithm

This section is from reference [Hall and Sriskandarajah, 1996]. The algorithm was first developed by Gilmore and Gomory [1964]. The algorithm described below solves the two-machine blocking flowshop with makespan minimization problem. The intuition behind the algorithm is that, ideally, the shortest processing time on machine 1 would be concurrent with that on machine 2, similarly for the second shortest processing times on the two machines, and so on. If this is not possible, a dual improvement step moves the current schedule towards feasibility at minimum cost. Let $\Phi(j)$ denote the job that follow job *j* in the sequence found.

Algorithm (Gilmore and Gomory 1964)

Step 1 Number the jobs such that $p_{2,j} \leq p_{2,j+1}, j = 1, \ldots, n - 1$. Initialize $G_1 = G_2 = \emptyset$.

Step 2 Find a function $\phi(j), j = 1, \ldots, n$, such that $p_{1,\phi(j)} \leq p_{1,\phi(j+1)}, j = 1, \ldots, n - 1$.

Step 3 Define a graph with *n* nodes (each representing a job) and no edges. The lengths $C_{j,j+1}$ of edges $(j, j + 1), j = 1, \ldots, n - 1$ that may be added later are given by $C_{j,j+1} = \max \{0, (\min \{p_{2,j+1} \leq p_{1,\phi(j+1)}\} - \max\{p_{2,j}, p_{1,\phi(j)}\})\}$ for $j = 1, \ldots, n - 1$.

Step 4 Set $j = 1$.

Step 4.1 If the undirected edge $(j, \phi(j))$ is not in the graph and $j \neq \phi(j)$, add it.

Set $j = j + 1$.

Step 4.2 If $j \leq n$, go to step 4.1.

Step 5 If the graph has only one connected component, go to Step 7. Otherwise, let k = argmin $\{C_{j,j+1} \mid j$ and $j+1$ are in different components$\}$, breaking ties arbitrarily.

Step 6 Add the undirected edge $(k, k+1)$ to the graph. If $p_{1,\phi(k)} \geq p_{2,k}$, set $G_1 = G_1 \cup \{(k, k+1)\}$. Otherwise set $G_2 = G_2 \cup \{(k, k+1)\}$. Go to Step 5.

Step 7 If $G_1 = \emptyset$, let $s = 0$. Otherwise, let the elements of G_1 be $\{(r_1, r_1 + 1), \ldots, (r_s, r_s + 1)\}$, where $r_1 \geq \ldots \geq r_s$.

Step 7.1 If $G_2 = \emptyset$, let $t = 0$. Otherwise, let the elements of G_2 be $\{(k_1, k_1 + 1), \ldots, (k_t, k_t + 1)\}$, where $k_1 \leq \ldots \leq k_t$.

Step 8 Define for $1 \leq e, g, h \leq n$ a function $\alpha_{e,g}(h)$ as follows: $\alpha_{e,g}(e) = g$, $\alpha_{e,g}(g) = e$, and $\alpha_{e,g}(h) = h$ if $h \neq e, g$. Set $j = 1$.

Step 8.1 If $t = s = 0$, set $\Phi(k) = \phi(k)$, $k = 1, \ldots, n$, and stop.

Step 8.2 Set $y = j$. If $t = 0$, set $i = s$ and go to Step 8.5. Otherwise, set $i = t$.

Step 8.3 Set $y = \alpha_{k_i, k_i+1}(y)$ and $i = i - 1$.

Step 8.4 If $i \geq 1$, go to Step 8.3. Otherwise, if $s = 0$, go to Step 8.6. Otherwise, set $i = s$.

Step 8.5 Set $y = \alpha_{r_i, r_i+1}(y)$ and $i = i - 1$.

Step 8.6 If $i \geq 1$, go to Step 8.5. Otherwise, set $\Phi(j) = \phi(y)$ and $j = j + 1$.

Step 8.7 If $j \leq n$, go to Step 8.2. Otherwise, stop.

This algorithm can be implemented in $O(n \log n)$ time. Now consider the following example.

Example

Job j	1	2	3	4	5	6	7	8
$P_{1,j}$	10	12	3	5	6	11	9	4
$p_{2,j}$	7	8	2	3	9	12	13	6

Step 1 gives:

Job j	1	2	3	4	5	6	7	8
$p_{2,j}$	2	3	6	7	8	9	12	13
$p_{1,j}$	3	5	4	10	12	6	11	9

Step 2 and 3 give:

Job j	$p_{2,j}$	$p_{1,\phi(j)}$	$\phi(j)$	$\max\{p_{2,j}, p_{1,\phi(j)}\}$	$\min\{p_{2,j}, p_{1,\phi(j)}\}$	$C_{j,j+1}$
1	2	3	1	3	2	0
2	3	4	3	4	3	1
3	6	5	2	6	5	0
4	7	6	6	7	6	1
5	8	9	8	9	8	0
6	9	10	4	10	9	1
7	12	11	7	12	11	0
8	13	12	5	13	12	-

At Step 4, the edges in the graph are (2, 3), (4, 6), and (5, 8).

At Step 5, the graph has components {1}, {2, 3}, {4, 6}, {5, 8}, and {7}.

At Step 6, edges (1, 2), (3, 4), (5, 6), and (7, 8) are added, $G_1 = \{(1, 2), (5, 6)\}$, $G_2 = \{(3, 4), (7, 8)\}$.

At Step 7, $r_1 = 5$, $r_2 = 1$, $k_1 = 3$, $k_2 = 7$.

At Step 8, for j = 1 we have: $y = \alpha_{7,8}(1) = 1$, $\alpha_{3,4}(1) = 1$, $\alpha_{1,2}(1) = 2$, $\alpha_{5,6}(2) = 2$, $\Phi(1) = \phi(2) = 3$. The steps are similar for $j = 2, \ldots, 8$.

The optimal sequence, with a makespan of 64, is given by:

Job j	1	2	3	4	5	6	7	8
$\Phi^*(j)$	3	1	6	2	4	8	5	7

Appendix B: Case Study Data

Line 1

	Part Number	Batch Size	P1 (s)	P2 (s)	Setup1 (s)	Setup2 (s)
1	187620A-05	8	25	30	180	180
2	185652C-01	77	94	75	180	180
3	187128A-01	55	113	111	180	180
4	187801A-01	80	25	36	180	180
5	184946C-02	26	78	54	180	180
6	185030F-01	33	102	83	180	180
7	186623B-05	7	67	163	180	180
8	183084G-05	14	55	63	180	180
9	185453D-02	60	98	125	180	180
10	186104C-01	77	50	39	180	180
11	184359B-01	169	119	149	180	180
12	183087H-03	14	61	60	180	180
13	183087H-04	22	155	183	180	180
14	185608B-01	45	33	51	180	180
15	181525K-01	85	46	35	180	180
16	186914B-01	12	80	80	180	180
17	184164F-02	51	48	59	180	180
18	184164G-01	50	68	87	180	180
19	181925C-11	50	54	75	180	180
20	181500F-01	90	73	90	180	180

Makespan according to NI scheduling (as sequenced) is 26.4 hours.

Optimal Makespan is 24.7 hours.

Optimal sequence:

1 – 20 – 5 – 17 – 13 – 3 – 18 – 12 – 14 – 11 – 2 – 19 – 10 – 4 – 9 – 16 – 7 – 6 – 8 - 15

Improvement 6.5%

Line 2

	Part Number	Batch Size	P1	P2	Setup1	Setup2
1	184177D-01	105	40	39	180	180
2	182481D-01	83	63	51	180	180
3	186695B-01	15	54	71	180	180
4	184466C-01	30	115	91	180	180
5	183550B-01	108	44	48	180	180
6	183262F-01	103	70	63	180	180
7	182880H-01	100	45	51	180	180
8	182750D-01	36	101	30	180	180
9	183381E-01	53	69	44	180	180
10	160461A-01	10	132	143	180	180
11	182880H-01	70	49	54	180	180
12	184182B-01	26	61	68	180	180
13	183262F-01	84	70	63	180	180
14	182435G-01	77	53	45	180	180
15	182887C-01	165	48	56	180	180
16	182459G-05	79	83	65	180	180
17	183442D-01	51	62	73	180	180
18	182770K-01	26	98	47	180	180
19	182465H-01	40	60	45	180	180
20	183442D-01	37	55	68	180	180

Makespan according to NI scheduling (as sequenced) is 23.7 hours.

Optimal Makespan is 22.3 hours.

Optimal sequence:

5 – 19 – 3 – 18 – 20 – 16 – 1 – 7 – 2 – 12 – 6 – 10 – 4 – 14 – 11 – 9 – 15 – 13 – 17 - 8

Improvement 6.1%

Line 3

	Part Number	Batch Size	P1	P2	Setup1	Setup2
1	183628E-01	30	64	50	180	180
2	183628E-01	40	94	47	180	180
3	183628E-01	40	64	50	180	180
4	183628E-02	100	74	56	180	180
5	183628E-02	100	74	40	180	180
6	183628E-02	100	74	76	180	180
7	183628E-02	125	104	86	180	180
8	183873A-02	72	36	49	180	180
9	183884C-01	20	43	52	180	180
10	184435A-01	80	61	93	180	180
11	184436B-01	60	56	46	180	180
12	184438A-01	216	77	41	180	180
13	184674C-01	34	51	81	180	180
14	184723B-01	26	69	44	180	180
15	184726C-01	100	76	46	180	180
16	185151A-01	120	54	71	180	180
17	185152B-01	70	58	54	180	180
18	185715A-01	20	78	41	180	180
19	185849A-01	10	87	78	180	180
20	186385B-01	20	63	78	180	180

Makespan according to NI scheduling (as sequenced) is 29.8 hours.

Optimal Makespan is 28.1 hours.

The optimal sequence is:

8 - 4 - 1 - 17 - 3 - 11 - 16 - 12 - 13 - 2 - 6 - 18 - 9 - 15 - 20 - 19 - 14 - 10 - 7 - 5

Improvement 5.7%

Line 4

	Part Number	Batch Size	P1	P2	Setup1	Setup2
1	181460-01C	32	33	35	180	180
2	181525K-02	50	38	48	180	180
3	181700J-01	80	109	95	180	180
4	181925C-01	132	37	36	180	180
5	181925C-11	120	40	34	180	180
6	181925C-12	40	39	47	180	180
7	182346E-01	40	138	85	180	180
8	182348C-11	40	78	55	180	180
9	182368F-01	46	70	63	180	180
10	182368F-01	33	81	93	180	180
11	182610D-01	61	139	79	180	180
12	182610D-02	16	89	59	180	180
13	182610D-03	12	44	62	180	180
14	182685H-01	175	68	80	180	180
15	183087G-06	20	67	55	180	180
16	184366D-01	5	74	87	180	180
17	186135C-01	35	87	91	180	180
18	186136C-01	24	67	83	180	180
19	186138D-02	24	37	56	180	180
20	186131F-02	20	89	104	180	180

Makespan according to NI scheduling (as sequenced) is 21.3 hours.

Optimal Makespan is 19.8 hours.

The optimal sequence is:

1 - 17 - 3 - 10 - 7 - 16 - 12 - 13 - 15 - 6 - 20 – 11 - 14 - 9 - 18 - 8 - 2 - 4 - 19 - 5

Improvement 7.1%

Bibliography

[1] Agnetis, A., D. Pacciarelli and F. Rossi, Batch scheduling in a two-machine flow shop with limited buffer, *Discrete Applied Mathematics*, v 72, p 234-260, 1997.

[2] Ahmadi, R.H. and P. and Kouvelis, Design of electronic assembly lines: An analytical framework and its application, *European Journal of Operational Research*, v 115, p 113-137, 1999.

[3] Ahuja, R., T. Magnanti and J. Orlin, *Network Flows, Theory, Algorithms, and Applications*, NJ: Prentice Hall, 1993.

[4] Ammons, J.C., M. Carlyle, L. Cranmer et al., Component allocation to balance workload in printed circuit card assembly systems, *IIE Transactions*, v 29, p 265-275, 1997.

[5] Bloat, A., Sequencing jobs for an automated manufacturing module with buffer, *European Journal of Operational Research*, v 96, p622-635, 1997.

[6] Brah, S. and A. Loo, Heuristics for scheduling in a flow shop with multiple processors, *Eur J Oper Res*, v 113 n 1, p 113-122, 1999.

[7] Caraffa, V. and S. Ianes, Minimizing makespan in a blocking flowshop using genetic algorithms, International Journal of Production Economics, v 70 n 2, p 101-115, 2001.

[8] Chakravarthy, K. and C. Rajendran, Heuristic for scheduling in a flowshop with the bicriteria of makespan and maximum tardiness minimization, *Production Planning and Control*, v 10 n 7, p 707-714, 1999.

[9] Cheng, T.C. G. Wang, Scheduling the fabrication and assembly of components in a two-machine flowshop, *IIE Transactions*, v 31, n 2, p 135-143, 1999.

[10] Chung, S. and D. Liao, Scheduling flexible flow shops with no setup effects, *IEEE Trans Rob Autom*, v 10 n 2, p 112-122, 1994.

[11] Coffman, E., M. Garey and D. Johnson, An application of bin-packing to multiprocessor scheduling, *SIAM Journal of Computing*, v 7, p 1-17, 1978.

[12] Crama, Y, J. Klundert and F. Spieksma, Production planning problems in printed circuit board assembly, *Discrete Applied Mathematics*, v 123, p 339-361, 2002.

[13] Guenther, H.O., M. Gronalt and R. Zeller, Job sequencing and component set-up on a surface mount placement machine, *Production Planning and Control*, v 9, n 2, p 201-211, 1998.

[14] Ellis, K., F. Vittes and J. Kobza, Optimization the performance of a surface mount placement machine, *IEEE transactions on Electronics Packaging Manufacturing*, v. 24, n 3, p 160-170, 2001.

[15] Garey, M., D. Johnson and R. Sethi, The complexity of flowshop and jobshop scheduling, *Mathematics of Operations Research*, v 1, p 117-129, 1976.

[16] Gilmore, P.C. and R.E. Gomory, Sequencing a one state-variable machine: a solvable case of the travelling salesman problem, *Operations Research*, v 12, p 655-679, 1964.

[17] Gupta, J. and V. Neppalli, Minimizing Total Flow Time in a Two-machine Flowshop Problem with Minimum Makespan, *International Journal of Production Economics*, v 69 n 3, p 323-338, 2001.

[18] Hall, L., Approximation algorithms for scheduling, in D. Hochbaum, *Approximation Algorithms for NP-hard Problems*, PWS Publishing Company, Boston, p 1-45, 1997.

[19] Hall, L., Approximability of Flow Shop Scheduling, *Mathematical Programming*, v 82, p 175-190, 1998.

[20] Hall, N. and C. Sriskandarajah, A survey of machine scheduling problems with bloking and no-wait in process, *Operations Research*, v 44, n 3, p 510-525, 1996.

[21] Hardin, J., *Resource-Constrained Scheduling and Production Planning: Linear Programming-Based Studies*, PhD thesis, Georgia Institute of Technology, 2001.

[22] Hong, T. and C. Wang, Heuristic Gupta-based flexible flow-shop scheduling algorithm, *Proc IEEE Int Conf Syst Man Cybern*, v 1, p 319-322, 2000.

[23] Johnson, S.M., Optimal two- and three-stage production schedules with setup times included, *Naval Research Logistics Quarterly*, v 1, p 61-68, 1954.

[24] Kleinau, U., Two-machine shop scheduling problems with batch processing, *Mathematical and Computer Modeling*, v 17, p 55-66, 1993.

[25] Lancia, G., Scheduling jobs with release dates and tails on two unrelated parallel machines to minimize the makespan, *European Journal of Operational Research*, v 120 n 2, p 277-288, 2000.

[26] Lee, C and L. Lei, Current Trends in Deterministic Scheduling, *Annals of Operations Research*, v 70, p 1-41, 1997.

[27] Leisten, R., Flowshop sequencing problems with limited buffer storage. *International Journal of Production Research*, v 28, p 2085- 2100, 1990.

[28] Lenstra, J. and D. Shmoys, Approximation algorithms for scheduling unrelated parallel machines, *Mathematical Programming*, v 46, p 259-271, 1990.

[29] Leon, V. J. and B. A. Peters, A comparison of setup strategies for printed circuit board assembly, *Computers & Industrial Engineering*, v 34, p 219-34, 1998.

[30] Liu, C. and S. Chang, Scheduling flexible flow shops with sequence-dependent setup effects, *IEEE Trans Rob Autom*, v 16 n 4, p 408-419, 2000.

[31] McGinnis, L.F., J.C. Ammons, M. Carlyle et al., Automated process planning for printed circuit card assembly, *IIE Transactions*, v 24, p 18-30,1992.

[32] Nakahara, H., PCB output 1998, *Printed Circuit Fabrication*, June 1999.

[33] Nemhauser, G. and L. Wolsey, *Integer and Combinatorial Optimization*, New York: Wiley, 1999.

[34] Noman, B, Scheduling flowshops with finite buffers and sequence-dependent setup times, *Computers and Industrial Engineering*, v 36, p 163-177, 1999.

[35] Nowicki, E., The permutation flow shop with buffers: a tabu search approach, *European Journal of Operational Research*, v 116, p 205-219, 1999.

[36] Palm, R., Reducing setup time for printed circuit assembly, *Hewlett-Packard Journal*, August 1996.

[37] Papadimitriou, C.H. and P.C. Kanellakis, Flowshop scheduling with limited temporary storage, *Journal of ACM*, v 27, p 533-549, 1980.

[38] Pinedo, M., *Scheduling: Theory, Algorithms, and Systems*, NJ: Prentice Hall, 1995.

[39] Potts, C. and L. Wassenhove, Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity, *Journal of the Operational Research Society*, v 43, p 395-406, 1992.

[40] Reeves, C., *Modern Heuristic Technique for Combinatorial Problems,* Oxford: Blackwell Scientific, 1993.

[41] Rios-Mercado, R. and J. Bard, Enhanced TSP-based heuristic for makespan minimization in a flow shop with setup times, *Journal of Heuristics*, v 5 n 1, p 53-70, 1999.

[42] Rossetti, M.D. and K. Stanford, Group sequencing a PCB assembly system via an expected sequence dependent setup heuristic, *Computers and Industrial Engineering*, v 45, n 1, p 231-254, 2003.

[43] Rowland, R., Rapid setup and changeover, *Surface Mount Technology*, July, 2003.

[44] Ruiz-Torres, A., E. Enscore and R. Barton, Parallel machine scheduling for minimizing the makespan and the average flow-time, *Industrial Engineering Research - Conference Proceedings*, Norcross GA USA, p 186-191, 1997.

[45] Shapiro, J., Mathematical Programming Models and Methods for Production Planning and Scheduling, in S. Graves, *Handbooks of Operations Research and Management Science*, v 4: *Logistics of Production and Inventory*, North Holland, p 371-443, 1993.

[46] Sule, D., Sequencing n jobs on two machines with setup, processing and removal times separated, *Naval Research Logistics Quarterly*, v 29, n 3, p 517-519, 1982.

[47] Tadei, R. and J. Gupta, Minimizing makespan in the two-machine flow-shop with release times, *Journal of the Operational Research Society*, v 49, n 1, p 77-85, 1998.

[48] Uetake, T. and H. Tsubone, Production scheduling system in a hybrid flow shop, *International Journal of Production Economics*, v 41 n 1-3, p 395, 1995.

[49] Vakharia, A.J. and B. Catay, Integrating family formation and scheduling in PCB manufacturing, *Annual Meeting of the Decision Sciences Institute*, v 3, p 1222, 1998.

[50] Weng, M., Scheduling flow-shops with limited buffer space, *Proceeding of the 2000 Winter Simulation Conference*, p 1359-1363, 2000.

[51] Wittrock, R., Scheduling Algorithms for Flexible Flow Lines, *IBM Journal of Research Development*, v 29, p 401-412, 1985.

[52] Williamson, D, L. Hall and J. Hoogeveen, Short Shop Schedules, *Operations Research*, v 45, p 288-294, 1997.

[53] Wolsey, L., *Integer Programming*, New York: Wiley, 1998.

[54] Yoshida, T. and K. Hitomi, Optimal two-stage production scheduling with setup times separated, *AIIE Transactions*, v 11, n 3, p 261-263, 1979.