MACHINE SCHEDULING WITH PREVENTIVE MAINTENANCES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

SAKİNE BATUN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
INDUSTRIAL ENGINEERING

JUNE 2006

Approval of the Graduate School of Natural and Applied Sciences

_____

Prof. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Prof. Çağlar Güven
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____

Prof. Meral Azizoğlu
Supervisor

Examining Committee Members

Prof. Ömer Kırca	(METU, IE)	_____
Prof. Meral Azizoğlu	(METU, IE)	_____
Assoc. Prof. Canan Sepil	(METU, IE)	_____
Asst. Prof. Ayten Türkcan	(METU, IE)	_____
Asst. Prof. M. Rüştü Taner	(Bilkent Uni., IE)	_____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Sakine BATUN

Signature :

ABSTRACT

MACHINE SCHEDULING WITH PREVENTIVE MAINTENANCES

BATUN, Sakine

M. Sc. Thesis, Department of Industrial Engineering
Supervisor: Prof. Meral Azizoğlu

June 2006, 104 pages

In manufacturing environments, machines are usually subject to down periods due to various reasons such as preventive maintenance activities, pre-accepted jobs and pre-known material shortages. Among these reasons, preventive maintenance, which is defined as the pre-planned maintenance activities to keep the machine in its operating state, has gained much more importance in recent years.

In this thesis, we consider the single machine total flow time problem where the jobs are non-resumable and the machine is subject to preventive maintenance activities of known starting times and durations. We propose a number of optimality properties together with the upper and lower bounding procedures. Using these mechanisms, we build a branch and bound algorithm to find the optimal solution of the problem. Our extensive computational study on randomly generated test instances shows that our algorithm can solve large-sized problem instances with up to 80 jobs in reasonable times.

We also study a two-alternative maintenance planning problem with minor and major maintenances. We give an optimizing algorithm to find the timing of the maintenances, when the job sequence is fixed.

Keywords: Preventive Maintenance, Machine Scheduling, Branch and Bound Algorithm

ÖZ

ÖNLEYİCİ BAKIM VARLIĞINDA MAKİNE ÇİZELGELEME

BATUN, Sakine

Yüksek Lisans Tezi, Endüstri Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Meral Azizoğlu

Haziran 2006, 104 sayfa

İmalat ortamlarında, önleyici bakım etkinlikleri, önceden kabul edilmiş işler ve önceden bilinen malzeme yetersizliği gibi nedenler, makinelerin işlememesini gerektirebilir. Bu nedenler arasında, makineleri işler durumda tutan önceden planlanmış bakım etkinlikleri olarak tanımlanan önleyici bakım etkinlikleri, özellikle son yıllarda daha çok önem kazanmıştır.

Bu tezde, önleyici bakım başlama zamanları ve süreleri belirlenmiş olan tek bir makinede toplam akış zamanını azaltmak için bölünemeyen işlerin çizelgelenmesi problemini ele almaktayız. En iyi çözüme ait özellikler, en iyi akış zamanı için üst ve alt sınır bulma yöntemleri, ve bu mekanizmaları kullanan bir dal-sınır algoritması önermekteyiz. Algoritmamızın ortalama ve en kötü performanslarını gözlemlemek için rassal test problemleri kullanarak gerçekleştirmiş olduğumuz kapsamlı deney sonuçları, 80'e kadar iş içeren büyük boyutlu problemlerin makul süreler içinde çözülebildiğini göstermektedir.

Birincil ve ikincil bakım etkinlikleri içeren önleyici bakım problemini de ayrıca ele almaktayız. Verilen bir iş çizelgesi için, bakım zamanlarını belirlemek amacıyla bir eniyileme algoritması önermekteyiz.

Anahtar Kelimeler: Önleyici Bakım, Makine Çizelgeleme, Dal-Sınır Algoritması

To my family

ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere appreciation to my supervisor Prof. Meral Azizoğlu for her wonderful guidance and endless support. She always untiringly and kindly contributed her time and knowledge to me with enthusiasm. No words can describe how grateful I am to her for what she added to my life and how happy I am to have the opportunity to work with such a nice person and a great researcher.

I would like to thank Prof. Ömer Kırca, Assoc. Prof. Canan Sepil, Asst. Prof. Mehmet Rüştü Taner and Asst. Prof. Ayten Türkcan for them being in my examining committee and their valuable contributions to this study. I am also thankful to Prof. Sencer Yeralan for his kind interest and helpful comments.

I would present my special thanks to my dear family. I am deeply indebted to my parents Ahmet Batun and Ayşe Batun, and my sister Betül Batun for their unconditional love and encouragement throughout not only this study but also my whole life. Their presence is the most precious blessing thing in my life and what I say remains absolutely missing as I cannot express my love and gratitude to them in words.

It was a really great pleasure for me to be a part of METU-IE assistants and I would like to thank all my friends at the department very much. My times while being an assistant would not be so pleasurable without their being and the completion of this study would never be possible without their technical and moral support. I am especially grateful to Zeynep Kirkizoğlu and Melih Özlen for their invaluable friendship and for what they brought to my life. It is really very hard to express how much they mean to me.

I also would like to thank my beloved friends Emine Seçkin Bozgüney, Şafak Sezginer, Erkut Sönmez, Mehmet Günhan Ertosun, Funda Karademir, Işıl Kirkizoğlu and Esra Kuyumcu for their existence as they always beautified my life and motivated me throughout this study.

I am grateful to all my professors, and I especially would like to extend my special thanks to Prof. Sinan Kayalıgil, Prof. Murat Köksalan and Asst. Prof. Haldun Süral for what they taught me and what they added to my personality. Their enthusiasm of making research and transferring knowledge was always very inspiring for me.

I am also thankful to the prospective METU-IE 2007 graduates for their warmth, understanding and kindness. My assistantship at the department would not be such an enjoyable experience without them.

Finally, I would like to thank TÜBİTAK (The Scientific and Technological Research Council of Turkey) for supporting this study through a graduate study scholarship.

TABLE OF CONTENTS

LIST OF TABLES

TABLE

LIST OF FIGURES

FIGURE

CHAPTER 1

INTRODUCTION

Scheduling is the allocation of scarce resources to activities over time. The resources take form of the machines and the activities take form of the jobs in a generic manufacturing environment. Machine scheduling is defined as the assignment of jobs to machines and determination of the sequence and processing start times of the jobs on the machines to achieve a goal or optimize a performance criterion. Due to its practical importance, there is enormous amount of research in many kinds of machine scheduling problems.

The majority of the studies in machine scheduling literature assume that the machines are continuously available. However, in real life applications, there exist scheduled unavailability periods on the machines due to various reasons such as preventive maintenance activities, pre-scheduled jobs and pre-known material shortages. Among these reasons, preventive maintenance is the most prevailing one due to its increased recognition and importance in today's competitive world for long-term survival.

In manufacturing environments, maintenance of a machine is defined as "all activities necessary to restore the machine to, or keep it in, a specified operating condition" (Pintelon and Gelders, 1992). By definition, there are two types of maintenance activities: the corrective maintenance which is carried out after breakdown to put the machine into its operating state, and the preventive

maintenance which is planned maintenance designed to keep the machine in its operating state. Preventive maintenance activities include lubrication, cleaning, adjusting, tool changeovers, replacement of machine parts, and inspection. The main purpose of these activities is to improve the machine life, increase the reliability, and hence reduce the frequency of corrective maintenance activities. Avoiding from corrective maintenance is an important issue as it is more costly and has more drastic consequences than preventive maintenance. This is due to the fact that, the corrective maintenance arises immediately resulting in unexpected need for maintenance resources, and repairing a machine usually takes longer time when compared to maintaining its operating state.

An effective preventive maintenance program reduces the frequency of machine breakdowns, thus the duration of the production downtime significantly. This results in increased availability of the machines, and hence increased production rate. Worsham (2000) points out the extensive amount of savings that can be brought by the reduced production downtime with the following notable example, taken from Newbrough (1967).

> "Downtime in an automobile plant assembly line at one time cost $10,000 per minute. Relating this to lost production time an automobile manufacturer reported that the establishment of a preventive maintenance program in their 16 assembly plants reduced downtime from 300 hours per year to 25 hours per year. With results such as this no well-managed plant can afford not to develop a preventive maintenance program."

Another benefit that could be gained from preventive maintenance is the increased expected life, hence well-timed replacement of the machines. Increased life and reduced breakdowns imply better overall machine condition that reduces the frequency of rework and scrap and improves the quality of the production and products. Besides increasing the production quantity and improving the production quality, preventive maintenance aims to maintain safe machine conditions for operators and prevent environmental damages.

In recent years, the increased complexity and highly intensive capital investment required for the implementation of the manufacturing systems, have added to the importance of the preventive maintenance activities. Clearly, a consequence of the complex systems combining "automation, integration and

flexibility" is the increased rate of failure occurrences (Simeu-Abazi and Sassine, 2001). More than that, it is very expensive, to keep machines down due to the increased capital intensiveness of the equipment, and setting tighter due dates owing to the increased value of customer satisfaction (Pintelon and Gelders, 1992). All of these issues together arise a strong need for an effective maintenance policy so as to increase the reliability and availability of the equipment.

Establishment of an effective maintenance policy that trade-offs the benefits and costs of the maintenance activities is a tactical level decision. There are several maintenance policies in the academic research literature and in manufacturing industry that can be classified into three main categories: purely corrective, purely preventive and mixed maintenance policies (Simeu-Abazi and Sassine, 2001).

Scheduling of the maintenance activities of the pre-set policy and scheduling of the jobs on the machines having pre-planned maintenance activities are operational level decisions. These two decisions can be made sequentially or simultaneously depending on the flexibility of the maintenance start times and durations.

Our research is concerned with the operational level decision problem of scheduling jobs on the machines that are subject to preventive maintenance activities with known start times and durations. We mainly focus on the single machine total flow time problem where the jobs are non-resumable.

We formulate this NP-hard problem as a mixed integer linear program and propose some properties of an optimal solution. Afterwards, we present a branch and bound algorithm that employs powerful bounds together with the optimality properties.

This thesis has seven chapters that altogether give a detailed account of machine scheduling work with preventive maintenances.

In Chapter 2, we present the formal definition of our problem together with the underlying assumptions and the classification scheme. We then, give the mathematical representation of the problem.

We present a brief review of the related literature in Chapter 3. The related studies are classified according to the machine environment and the performance criterion considered.

We discuss our solution approach in Chapter 4. We present our branch and bound algorithm in detail after discussing the properties of an optimal solution, and upper and lower bounding procedures.

In Chapter 5, we present the results of our extensive computational experiment. We first discuss the parameters used in generating the problem instances and present the performance measures used to evaluate the effectiveness of our bounding procedures and branch and bound algorithm. The computational experiments involve testing the effects of the reduction mechanisms and bounding procedures and the problem parameters on the performances of our branch and bound algorithm.

We also address a maintenance activities scheduling problem with two discrete alternatives at each decision point, and present the work in Chapter 6.

Finally, Chapter 7 concludes our study with a few ideas on the possible future research directions.

CHAPTER 2

PROBLEM DEFINITION

In this chapter, we first provide a formal presentation of our problem with its underlying assumptions and then the classification scheme used throughout this study. Finally, we present the mathematical models of our problems.

2.1 Problem Statement

Machine scheduling problems with availability constraints arise due to the several reasons in manufacturing environments. Preventive maintenance is one of the main reasons of machine unavailability from practical point of view, hence we use the terms “maintenance”, “preventive maintenance” and “unavailability” interchangeably throughout the thesis.

We consider the flow time problem on a single machine that is subject to q unavailability periods whose starting times (s_k) and durations (t_k) are known in advance. There are n jobs to be scheduled on the machine and p_j denotes the processing requirement of job j. The jobs are non-resumable, i.e., they should be processed without any interruption, in other words, the jobs whose processing is interrupted by a maintenance activity must be restarted.

All jobs are ready at the beginning of the planning horizon, i.e., their release times are zero, so the total flow time and total completion time objectives are

equivalent. Therefore, “flow time” and “completion time” are used interchangeably throughout the study.

A batch is the group of jobs scheduled to be processed in the same availability period. The availability period just before the k^{th} maintenance task is the k^{th} availability period and the corresponding batch is called batch k, B_k.

Throughout this study, we make the following additional assumptions:

- The preventive maintenance activities are sufficient to keep the machine in its operating state without any failures, i.e., there is no corrective maintenance activity.
- There can be at most one job being processed on the machine and the machine cannot process any job while it is being maintained.
- The maintenance period, i.e., the duration between two consecutive maintenance tasks, is no smaller than the maximum processing time to ensure that at least one job is processed in each batch.
- All parameters (processing times of the jobs, starting times and durations of maintenance tasks) are known with certainty and are not subject to change, i.e., the system is deterministic.
- All jobs are ready for processing at time zero, i.e., the system is static.

2.2 Classification Scheme

In this section, we present a classification scheme for machine scheduling problems with availability constraints by slightly modifying the schemes by Sanlaville and Schmidt (1998) and Schmidt (2000).

Our classification scheme includes three parameters, a, b and g, to categorize various scheduling problems. The three fields $a \mid b \mid g$, describe the characteristics of the problem environment.

a specifies the machine environment and the possible values for a are as listed below:

Single Stage Systems

$a = 1$: Single Machine

$a = \mathrm{P}$: Identical Parallel Machine

$a = \mathrm{Q}$: Uniform Parallel Machine

$a = \mathrm{R}$: Unrelated Parallel Machine

Multi Stage Systems

$a = \mathrm{F}$: Pure Flow Shop

$a = \mathrm{FF}$: Flexible (Hybrid) Flow Shop

$a = \mathrm{J}$: Pure Job Shop

$a = \mathrm{FJ}$: Flexible (Hybrid) Job Shop

$a = \mathrm{O}$: Open Shop

Special characteristics of jobs and machines and specific system restrictions, - like unavailability periods in our case- are placed in field b .

Parameter $b_1 \in \{\text{r-a, nr-a, sr-a}\}$ is used to describe the resumability characteristic of the jobs in the existence of unavailability periods.

When a job is to be preempted, either its processing is continued or must be completely or partially reprocessed after the machine becomes available. The possible values for b_1 are,

$b_1 = \text{r-a}$: The jobs are resumable, i.e., their processing are continued.

$b_1 = \text{nr-a}$: The jobs are non-resumable, i.e., their processing must be restarted.

There is also semi-resumability structure which is introduced by Lee (1999). A semi-resumable job ($b_1 = \text{sr-a}$) can be continued after the machine becomes available, but a portion of its processing before the unavailability period, must be restarted.

Some different patterns of availability periods are studied by various researchers and different classifications are made for availability period patterns. On single machines, a number of the studies consider only one unavailability period on a machine ($b_2 = \text{single}$) whereas the others consider more than one unavailability period ($b_2 = \text{multi}$). For multiple machines environment “single” or “multi” is used for b_2 only if the same structure holds on each machine. If the structure is different, then different notation is used depending on the machine

environment. For example, for the parallel machine environments with single maintenance on p machines and multiple maintenance periods on r machines single-p, multi-r is used as b_2 and for m-machine flow shop problem where there is one maintenance period on the machine k, $b_2 = \text{single-M}(k)$ is used.

Preventive maintenances are one of the activities that cause machine unavailability. In the parallel machine systems, there are two different unavailability period schemes depending on the maintenance resources (Lee and Chen, 2000). If there are sufficient maintenance resources, then any number of machines can be maintained simultaneously. This case is called independent case since maintenance of a machine does not depend on those of the others. In this case, we may assume that the maintenance resources are not limited. In the dependent case, maintenance resources are not sufficient, and only a limited number of machines can be maintained at the same time. Parameter $b_3 \in \{\text{dep-}k\}$ is used to describe dependent maintenance periods in a parallel machine system and it means that there can be at most k machines maintained at a time. A special example of the dependent case is the zigzag pattern where at most one machine is unavailable at a given time, i.e., k=1. Note that, $b_3 = \text{dep-1}$ and $b_3 = \text{zz}$ can be used interchangeably to denote the pattern is zigzag. b_3 field is kept empty if the maintenances are independent. b_3 can be used for multi stage problems as well.

The starting time of maintenance activities may be fixed and known in advance or they may be non-fixed, hence decision variables. Fixed or non-fixed status of the maintenance activities is described in field b_4. Accordingly,

$b_4 = \text{nfix-TW}$: There is a time window for the starting time of each maintenance period.

$b_4 = \text{nfix-P}$: The time between two consecutive maintenance periods, should not be longer than a specified time period.

Periodic maintenance activities can also be performed after a fixed interval of time after the completion of the last maintenance task. This pattern appears as $b_4 = \text{fix-P}$ which denoting an environment in which the machines are maintained at every certain time period.

Parameter $b_5 \in \{\varnothing, r_j\}$ describes release (ready) times of the jobs. All release times can be zero ($b_5 = \varnothing$) or job j may be released at time r_j ($b_5 = r_j$).

Another characteristic that is included in b field is the sequence dependence, i.e., precedence relation among the jobs. If one or more jobs have to be completed before another job can be started, then $b_6 = prec$ is used.

The last field g denotes the criterion/criteria to be optimized. Commonly used performance measures are:

$C_{\max}$: Makespan, i.e., the maximum job completion time

$\sum_j C_j$: Total completion time

$\sum_j w_j C_j$: Total weighted completion time

$\sum_j F_j$: Total flow time ($F_j = C_j - r_j$)

$\sum_j w_j F_j$: Total weighted flow time

$\sum_j L_j$: Total lateness ($L_j = C_j - d_j$)

$L_{\max}$: Maximum lateness

$\sum_j T_j$: Total tardiness ($T_j = \max\{0, C_j - d_j\}$)

$\sum_j w_j T_j$: Total weighted tardiness

$T_{\max}$: Maximum tardiness

$\sum_j E_j$: Total earliness ($E_j = \max\{0, d_j - C_j\}$)

$E_{\max}$: Maximum earliness

$\sum_j U_j$: Number of tardy jobs ($U_j = 1$ if $T_j > 0$, $U_j = 0$ if $T_j = 0$)

$\sum_j w_j U_j$: Weighted number of tardy jobs

According to this classification scheme, our problems can be represented as:

- 1|nr-a, multi, fix|$\sum C_j$: single machine, total completion time problem with non-resumable jobs and multi fixed maintenance tasks
- 1|nr-a, multi, fix-P|$\sum C_j$: single machine, total completion time problem with non-resumable jobs and multi fixed maintenance interval of same length, i.e., periodic maintenances.

2.3 Mathematical Formulation of Our Problems

In this section we give mixed integer linear programming models of the 1|nr-a, multi, fix|$\sum C_j$, 1|nr-a, multi, fix-P|$\sum C_j$ and 1|nr-a, single, fix|$\sum C_j$ problems.

We make use of the following property in our mathematical models:

Property 1. The jobs within each batch are ordered in nondecreasing order of their processing times, i.e., Shortest Processing Time (SPT) order, in all optimal solutions.

The result follows immediately from the optimality of the SPT rule for the total completion time objective when the machine is continuously available. (see Smith, 1956)

As we know the optimal sequence within each batch due to Property 1, our decision is only related with the assignment of jobs to the batches. We hereafter assume that the jobs are indexed in SPT order, i.e., $p_1 \le p_2 \le ... \le p_n$ and the maintenance tasks are indexed in increasing order of their starting times.

We now give the indices, parameters and decision variables that define our problems.

Indices

i, j : job indices, $1,\ldots,n$

k : maintenance task index $1,\ldots,q$

Parameters

s_k: starting time of k^{th} maintenance task (or k^{th} unavailability period)

$s_0 = 0, t_0 = 0$ since we assume that the planning horizon starts with an availability period.

t_k: duration of k^{th} maintenance task

p_j: processing time of job j

M_{jk}: an upper bound on the completion of job j in batch k

In our formulations, we use $M_{jk} = \min\left\{s_k, s_{k-1} + t_{k-1} + \sum_{i=1}^{j-1} p_i + p_j\right\}$. Note that s_k is an upper bound on the completion of the last job of batch k, $\sum_{i=1}^{j-1} p_i$ is the maximum possible processing before job j and $s_{k-1} + t_{k-1}$ is the start time of batch k.

When the maintenance durations, i.e., t_k s, and availability interval lengths, i.e., $s_k - s_{k-1} - t_{k-1}$ s, are fixed to t and T respectively, i.e. periodic case is considered, M_{jk} becomes $\min\left\{(k-1)t + kT, (k-1)(T+t) + \sum_{i=1}^{j-1} p_i + p_j\right\}$.

Decision Variables

$$x_{jk} = \begin{cases} 1 & \text{if job } j \text{ is scheduled in the } k^{th} \text{ availability period} \\ 0 & \text{otherwise} \end{cases}$$

C_j: completion time of job j

The Gantt chart given in Figure 2.1 illustrates the problem environment.

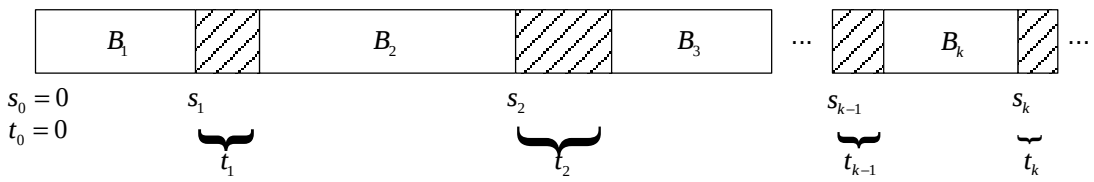


Figure 2.1 Gantt Chart Representation of Our Problem

2.3.1 The 1|nr-a, multi, fix|$\sum C_j$ Problem

$$\text{Min} \sum_{j=1}^{n} C_j \tag{2.1}$$

subject to

$$\sum_{k=1}^{q} x_{jk} = 1 \qquad \forall j \tag{2.2}$$

$$\sum_{j=1}^{n} p_j x_{jk} \le s_k - s_{k-1} - t_{k-1} \qquad \forall k \tag{2.3}$$

$$C_j \ge s_{k-1} + t_{k-1} + \sum_{i=1}^{j} p_i x_{ik} - M_{jk}(1 - x_{jk}) \qquad \forall j, \forall k \tag{2.4}$$

$$x_{jk} \in \{0,1\} \qquad \forall j, \forall k \tag{2.5}$$

The objective function of the model expressed in (2.1) is the sum of the job completion times. Constraint set (2.2) ensures that each job is assigned to one batch, i.e., scheduled to be processed in one availability period. Constraint set (2.3) guarantees that the total processing assigned to an availability period is not greater than its length. Constraint set (2.4) defines the completion time of each job. Note that C_j takes on value $s_{k-1} + t_{k-1} + \sum_{i=1}^{j} p_i x_{ik}$ when $x_{jk} = 1$, i.e., it is assigned to batch k. So we ignore the nonnegativity constraint on C_j. Finally, constraint set (2.5) sets up the binary restrictions for x_{jk} variables.

2.3.2 The 1|nr-a, multi, fix-P|$\sum C_j$ Problem

$$\text{Min} \sum_{j=1}^{n} C_j \tag{2.1}$$

subject to

$$\sum_{k=1}^{q} x_{jk} = 1 \qquad \forall j \tag{2.2}$$

$$\sum_{j=1}^{n} p_j x_{jk} \le T \qquad \forall k \qquad (2.6)$$

$$C_j \ge (k-1)(T+t) + \sum_{i=1}^{j} p_i x_{ik} - M_{jk}(1-x_{jk}) \qquad \forall j, \forall k \qquad (2.7)$$

$$x_{jk} \in \{0,1\} \qquad \forall j, \forall k \qquad (2.5)$$

The objective function (2.1) and the constraint sets (2.2) and (2.5) are explained in the mathematical model of the 1|nr-a, multi, fix| $\sum C_j$ problem.

Constraint set (2.6) guarantees that the total processing assigned to each batch is not greater than the maintenance period length. Constraint (2.7) defines the completion time of each job. Note that the completion time of job j is $(k-1)(T+t) + \sum_{i=1}^{j} p_i x_{ik}$ when processed in batch k.

When there is only one maintenance task, the problem becomes 1|nr-a, single, fix| $\sum C_j$. Assume the starting time of the single maintenance is s and its duration is t. We let x_j denote the binary decision variable indicating whether job j is processed before the maintenance task or after it. The mathematical formulation of the problem is provided below.

$$\text{Min} \sum_{j=1}^{n} C_j \qquad (2.1)$$

subject to

$$\sum_{j=1}^{n} p_j x_j \le s \qquad \forall k \qquad (2.8)$$

$$C_j \ge \sum_{i=1}^{j} p_i x_i - M_{j1}(1-x_j) \qquad \forall j \qquad (2.9)$$

$$C_j \ge s + t + \sum_{i=1}^{j} p_i (1-x_i) - M_{j2} x_j \qquad \forall j \qquad (2.10)$$

$$x_j \in \{0,1\} \qquad \forall j \qquad (2.11)$$

The objective function (2.1) is explained in the previous models. Constraint set (2.8) implies that the processing assigned before the maintenance task is not greater than the length of maintenance period. Constraint sets (2.9) and (2.10) define the completion times of the jobs where $M_{j1} = \min\left\{s, \sum_{i=1}^{j} p_i\right\}$ and $M_{j2} = s + t + \sum_{i=1}^{j} p_i$. Note that the completion time of job j is $\sum_{i=1}^{j} p_i x_i$ if processed before maintenance or $s + t + \sum_{i=1}^{j} p_i(1 - x_i)$ if processed after maintenance. Finally, constraint set (2.11) sets up the binary restrictions for x_j variables.

CHAPTER 3

THE REVIEW OF RELATED LITERATURE

In majority of the scheduling studies, the machines are assumed to be continuously available. However, in manufacturing practice, the resources are subject to down periods due to various planned resources like preventive maintenances, pre-accepted orders and pre-known material shortages. Having recognized the importance of pre-planned, nonproductive activities, the researchers have begun to forward their studies towards this fruitful area and preventive maintenance activities, in general machine availability constraints, have been the subject of many studies.

Sanlaville and Schmidt (1998) and Schmidt (2000) present thorough surveys of the studies with availability constraints and they suggest a well structured classification scheme to represent various availability patterns. Lee *et al.* (1997) review the literature on machine scheduling with availability constraints.

We review the related literature according to the machine environment as single and multi stage environments.

3.1 Single Stage Problems

Single stage, single machine and parallel machine, studies are reviewed according to the g field, i.e., objective function.

3.1.1 Single Machine Problems

The majority of single machine studies consider total flow time and makespan objectives. There are also some notable studies that minimize a due-date related objective.

a) Total Flow Time

Adiri *et al.* (1989) study the 1|nr-a, single, fix|$\sum C_j$ problem. They show that the problem is NP-hard in the ordinary sense and provide a worst case error bound of $\frac{1}{4}$ for the Shortest Processing Time (SPT) rule. They also study the stochastic version of the problem. Lee and Liman (1992) study the 1|nr-a, single, fix|$\sum C_j$ problem as well. They prefer to use the term "scheduled maintenance" in place of "unavailability period" as they find the former term more appropriate for representing the environment. They reduce the problem to the even-odd partition, thereby proving its binary NP-hardness. They present a problem instance with a tight error bound of $\frac{2}{7}$ thereby providing a counter example for the $\frac{1}{4}$ worst case bound of Adiri *et al.* (1989). Sadfi *et al.* (2005) study the same problem and propose a heuristic algorithm with a tight worst case error bound of $\frac{3}{17}$. Their algorithm first constructs a feasible schedule by using the SPT rule, then improves this initial schedule by exchanging the jobs before the maintenance with the ones scheduled after the maintenance. They test the relative performance of their heuristic with respect to the SPT rule and report that it performs satisfactorily well.

Lee (1996) presents a well structured review of the previous studies and includes new results for different machine environments with various performance measures. The author shows that the 1|r-a, single, fix|$\sum C_j$ problem can be solved optimally in polynomial time by the SPT rule.

Qi *et al.* (1999) show that the1|nr-a, multi, nfix-P|$\sum C_j$ problem is NP-hard in strong sense and propose heuristic algorithms that are mainly based on the SPT rule and the idea of using fewer batches. They provide some special conditions for

which these heuristics give optimal solutions. The authors propose some properties of an optimal solution and develop a branch and bound algorithm using these rules as reduction mechanisms. Their lower bound uses the SPT rule to find the minimum contribution of the processing times, and finds the maximum number of jobs that can be processed in the next batch to find the minimum contribution of the maintenance duration. Their computational experiments show the efficiency of their algorithms. Aktürk *et al.* (2003) consider the same problem in the context of flexible manufacturing systems scheduling and present similar theoretical results. They provide a mixed integer linear programming model and obtain exact solutions for small size problem instances. They propose several heuristic algorithms based on dispatching rules and local search to minimize the total completion time by minimizing the contribution of processing times and number of batches simultaneously. Their computational experiments show the satisfactory performance of their algorithms. Qi *et al.* (1999) and Aktürk *et al.* (2003, 2004) show that the SPT sequence is optimal if the number of batches used is one or two, or maintenance duration is zero. Aktürk *et al.* (2004) show that the worst case error bound of the SPT rule is 1 if the number of batches used is more than two.

Chen (2006a) studies a special case of the1|nr-a, multi, nfix-P|$\sum C_j$ problem. In his problem, there is a minimum allowable duration between two maintenance tasks and an upper limit on the maintenance period duration. He develops four different mixed binary integer programming models to solve small size problem instances and proposes a heuristic algorithm mainly based on the SPT rule to solve large size problems. Computational experiments report the relative performance of optimization models and show the efficiency of the heuristic algorithm.

Chen (2006c) considers the 1|nr-a, multi, fix-P|$\sum C_j$ problem with minimum number of batches. He proposes a number of theorems and builds a branch and bound algorithm utilizing these theorems. His branch and bound algorithm is capable of solving problem instances with up to 30 jobs. The lower bound is found by ignoring the succeeding maintenance activities and sequencing the unscheduled jobs in the SPT order. He develops a heuristic algorithm which is mainly based on constructing a minimum batch schedule and improving the

solution by the theorems. The results of his experiments under different settings reveal the satisfactory performance of the heuristic algorithm.

a) Total Weighted Flow Time

Lee (1996) shows that the 1|r-a, single, fix| $\sum w_j C_j$ problem is NP-hard in the ordinary sense even when $w_j=p_j$. He proposes a greedy heuristic algorithm called "combined algorithm" which is basically modified Weighted Shortest Processing Time (WSPT) rule. The worst case bounds of the WSPT rule and this algorithm can be arbitrarily large as the number of jobs approaches to infinity. The author shows that combined algorithm's worst case error bound is 1 and tight when $w_j=p_j$, whereas WSPT rule's error bound is still arbitrarily large. He builds a dynamic programming algorithm by establishing and using the optimality property that there exists an optimal solution such that the jobs finished before the unavailability period and the jobs finished after the unavailability period follow the WSPT order within themselves.

Lee (1996) also addresses the problems with non-resumable jobs. He points out that the 1|nr-a, single, fix| $\sum w_j C_j$ problem is NP-hard and shows that the error bound of the WSPT algorithm can be arbitrarily large even when the weights of the jobs are proportional to their processing times.

Graves and Lee (1999) study the scheduling of the maintenance activities and the semi-resumable jobs on a single machine to minimize total weighted completion times. The "semi-resumability" of job j is implied by an additional set-up of s_j units whenever it is resumed. The processing of a job interrupted due to the maintenance is restarted immediately after the maintenance ends, i.e., the jobs are not preempted by the others. The starting time of the maintenance is also a decision variable, the duration of maintenance is t units and there can be at most T time units between two consecutive maintenance periods. The authors examine the problem for two planning horizon scenarios. In the first scenario, the planning horizon is long compared to T and the total set up times and total processing times of all jobs are no more than $2T$. The authors develop optimality properties, and using one of those properties, they find that the number of maintenances to be performed is at most two. The authors show that the 1|sr-a, multi, nfix-P| $\sum w_j C_j$ problem is NP-

hard in the ordinary sense and develop a dynamic programming algorithm that runs in pseudo-polynomial time.

In the second scenario, some portion of T has already passed and the planning horizon is short compared to T. The total set up times and total processing times of all jobs are strictly greater than T' where T' is the remaining allowable time till the maintenance activity. So, one maintenance period must be scheduled in the horizon and the associated problem becomes 1|sr-a, single, nfix-TW|$\sum w_j C_j$ where time window for the starting time of the maintenance is [0, T']. The authors show that this problem is NP-hard and propose two different dynamic programming algorithms that run in pseudo-polynomial time. Moreover, they show that the SPT rule solves the 1|sr-a, single, nfix-TW|$\sum C_j$ problem when the maintenance is inserted at the latest possible time without splitting a job.

Wang *et al.* (2005) address the resumable single machine scheduling problem so as to minimize total weighted completion time. They prove that the 1|r-a, multi, fix|$\sum w_j C_j$ problem is NP-hard in the strong sense. They study the special cases with the proportional weights (1|r-a, multi, fix, w_j=p_j|$\sum w_j C_j$) and single availability period (1|r-a, single, fix|$\sum w_j C_j$). The authors show that the proportional weight case is NP-hard in the strong sense and Longest Processing Time (LPT) rule has a tight worst case error bound of 1. For the single availability constraint case, the authors propose a heuristic procedure that runs in $O(n^2)$ time and has a worst case error bound of 1. The idea of their heuristic is to improve the WSPT schedule by rescheduling the preempted job so that it is processed before the unavailability period. This heuristic outperforms the WSPT rule and the combined algorithm of Lee (1996) that have no worst case error bounds.

Kacem *et al.* (2006) study the 1|nr-a, single, fix|$\sum w_j C_j$ problem. They propose several lower bounds and their mathematical properties. They propose three methods to find the optimal solution: branch and bound algorithm, mixed integer programming model solutions and dynamic programming algorithm. Their computational results show that the performances of the branch and bound algorithm and dynamic programming method are compatible and satisfactorily well

(can solve problem instances with up to 3000 jobs), and these algorithms perform significantly better than the mixed integer programming model solutions.

c) Makespan

Any sequence is an optimal solution to the 1|r-a, single, fix|C_{max} problem. The 1|nr-a, single, fix|C_{max} and 1|nr-a, multi, fix|C_{max} problems are NP-hard in the strong sense (Lee, 1996). For the problem with single unavailability period, Lee (1996) shows that using LPT rule and assigning as many jobs as possible before the unavailability period has a worst case error bound of $\frac{1}{3}$ and this bound is tight.

Mauguiere *et al.* (2005) consider the single machine makespan problem with several maintenance tasks. They consider a job set having both resumable and non-resumable jobs that have release times and latency durations, i.e., the jobs are subject to some post operations after completion. Moreover, they suggest a new classification scheme for maintenance tasks as "*crossable*" and "*non-crossable*" periods. A crossable maintenance task allows resumability whereas a non-crossable one not. A job can be resumed only if it is resumable and is preempted by a crossable maintenance task. The authors consider several maintenance tasks on the machine including both crossable and non-crossable ones. They propose a branch and bound algorithm, test their algorithm under different problem settings and report that it performs extremely well.

d) Due Date Related Performance Criteria

Lee (1996) shows that the single machine, single fixed maintenance maximum lateness and number of tardy jobs problems are solved in polynomial time when the jobs are resumable. The associated problems are denoted as 1|r-a, single, fix|L_{max}, 1|r-a, single, fix|$\sum U_j$ and solved by the EDD rule and Moore Hodgson's algorithm respectively. However when the jobs are non-resumable, the associated 1|nr-a, single, fix|L_{max}, and 1|nr-a, single, fix|$\sum U_j$ problems become NP-hard. Lee (1996) shows that the L_{max} of the EDD sequence deviates from optimal L_{max} by at most p_{max} units, and Moore and Hodgson's algorithm gives at most one more tardy jobs than optimal number of tardy jobs.

As mentioned before Graves and Lee (1999) study scheduling maintenance activities and semi-resumable jobs on a single machine to minimize total weighted completion time. They also consider the maximum lateness problem. They propose a pseudo-polynomial dynamic programming algorithm for the 1|sr-a, multi, nfix-P, $|L_{max}$ problem and therefore show that the problem is NP-hard in the ordinary sense. When the planning horizon is short, hence implying one maintenance activity, the problem becomes 1|sr-a, single, nfix-P|L_{max}. For this problem, the authors show that there exists an optimal solution in which the jobs are ordered by the EDD rule and the maintenance is inserted as late as possible without splitting a job.

Liao and Chen (2003) consider the single machine scheduling problem with periodic maintenance activities and non-resumable jobs so as to minimize the maximum tardiness. The authors state that the 1|nr-a, multi, fix-P|T_{max} problem is NP-hard as the 1|nr-a, single, fix|L_{max} problem is NP-hard (Lee, 1996). They present a branch and bound algorithm whose efficiency is improved by dominance conditions and a lower bound. Although not stated explicitly, their dominance conditions can be generalized for all regular performance measures. Their lower bound uses the EDD order by ignoring the maintenances. To find an initial feasible solution to the branch and bound algorithm and an approximate solution for the problem, the authors develop a heuristic procedure. Their heuristic constructs an initial solution with minimum number of maintenance periods by using a bin packing algorithm and improves the resulting solution using the results of their dominance conditions. Their computational results, using several parameter settings, reveal the satisfactory behavior of the heuristic algorithm.

Chen (2006b) addresses special cases of the 1|r-a, multi, nfix-P|$\sum T_j$ and 1|nr-a, multi, nfix-P|$\sum T_j$ problems where a minimum allowable duration is imposed on the period between two consecutive maintenance tasks. He develops two different mixed binary integer programming models and discusses their relative performance by presenting computational results.

3.1.2 Parallel Machine Problems

a) Total Flow Time

Kaspi and Montreuil (1988) show that the identical machine total flow time problem is solved by SPT rule if there are arbitrary machine initial availability starting times.

Lee and Liman (1993) consider a special case of the P2|nr-a, single-1, fix|$\sum C_j$ problem in which one of the two parallel identical machines is not available after a specified time. The authors reduce the problem into an even-odd partition problem, thereby establishing its NP-hardness in the ordinary sense. They propose a pseudo-polynomial dynamic programming algorithm. They also propose a heuristic algorithm that is based on the SPT rule with a slight modification and show that the worst case error bound of this heuristic is $\frac{1}{2}$, and this bound is tight.

Mosheiov (1994) study the generalization of the Lee and Liman's (1993) problem to *m* identical parallel machines where some of the machines are not available after arbitrary specified times. Hence the problem is Pm|nr-a, single-*p*, fix|$\sum C_j$. The author proposes a lower bound for the two machine case, and extends it to the general *m*-machine case. His lower bound is mainly based on the SPT rule and splitting the last job on the capacitated machine if its processing should be interrupted due to the capacity constraint. Moreover, he improves and extends the heuristic of Lee and Liman (1993) that is based on the SPT rule, for the multi-machine case. He shows that the heuristic and the lower bound are asymptotically optimal as the number of jobs increases. The performance of the SPT-based heuristic and lower bound are tested under different settings and the results reveal their superior performance.

b) Total Weighted Flow Time

Lee (1996) shows that the P2|r-a, single, fix|$\sum w_j C_j$ and P2|nr-a, single, fix|$\sum w_j C_j$ problems are NP-hard. He proposes exact dynamic programming algorithms for the special cases where one machine is always available and there is

one maintenance task on the other machine, i.e., the P2|r-a, single-1, fix| $\sum w_j C_j$ and P2|nr-a, single-1, fix| $\sum w_j C_j$ problems.

Lee and Chen (2000) study the total weighted completion time problem on *m* identical parallel machines where each machine must be maintained once during the planning horizon [0, *T*] and the starting times of the maintenance activities, each having fixed duration of *t*, are also to be decided. They consider two cases: In the first case, they assume unlimited maintenance resources so that multiple machines can be maintained simultaneously. The second case is closer to practice and there are limited resources so that at most one machine can be maintained at a time. The authors show that both the Pm|nr-a, single, ind, nfix-TW | $\sum w_j C_j$ and Pm|nr-a, single, dep-1, nfix-TW | $\sum w_j C_j$ problems with a time window of [0, *T*], are NP-hard even when the weights of the jobs are equal. They use set partitioning type formulations to represent the problems and develop branch and bound algorithms to solve them. The linear relaxation of a set partitioning type problem using the column generation idea is used to find lower bounds. Their computational results show that the algorithms can solve problem instances up to 40 jobs and any number of machines in reasonable time. The results reveal that the satisfactory behavior of the algorithms is due to the small integrality gap at the root node leading to explore only few nodes.

c) Makespan

Lee (1991) considers a special case of the Pm|nr-a, single, fix|$C_{\max}$ problem where the machines are unavailable only at the beginning of the planning horizon. He proves that $C_{\max}$ value of the LPT rule is at most $\frac{3}{2}-\frac{1}{2m}$ times the optimal $C_{\max}$ value and this bound is tight. The author modifies LPT (MLPT) and gets an improved a worst case error bound of $\frac{1}{3}$. He also suggests a lower bound but Lee *et al.* (2000) show that it may not be a valid bound for a special case of the problem. Lin (1998) shows that the error bound of the MLPT is tight when there are three or more machines. For the two machine case, he shows that worst case error bound

is $\frac{1}{4}$, but not $\frac{1}{3}$. Chang and Hwang (1999) also consider the same problems as those in Lee (1991) and Lin (1998) and use a bin packing heuristic algorithm called MULTIFIT algorithm. They show that the $C_{\max}$ value of the algorithm is at most $\frac{9}{7}+2^{-k}$ times of the optimal $C_{\max}$ value where k is number of iterations used in the MULTIFIT algorithm.

Lee (1996) presents several important results, approximate and exact solution algorithms considering different performance measures for the parallel machine problems with a single maintenance task on each machine. As the author states, the Pm|r-a, single, fix|$C_{\max}$ problem is strongly NP-hard as the problem without availability constraints is strongly NP-hard. He shows that the error bound of the LPT rule can be arbitrarily large even when there are two-machines. He defines a special case of the problem where one of the machines is always available, finds worst case error bounds of the LPT and MLPT rules for the general m. He mentions that minimizing $C_{\max}$ is NP-hard when the jobs are non-resumable and shows that the makespan values of the list scheduling and LPT rule can be at most m and $\frac{m+1}{2}$ times of the optimal $C_{\max}$ value, respectively.

Suresh and Chaudhuri (1996) study the unrelated machines makespan problem with non-resumable jobs, i.e., the Rm|nr-a, multi, fix|$C_{\max}$ problem. Their lower bound takes the maintenance periods as jobs with release times, and then shifts them to their release times. Their upper bound uses the sequence of the lower bound and improves it by shifting or exchanging mechanisms. Their computational experience shows the satisfactory performances of their bounds.

Hwang and Chang (1998) consider the Pm|nr-a, single, dep, fix|$C_{\max}$ problem. They show that the LPT rule has a tight worst case error bound of 1 if no more than $\frac{m}{2}$ can be maintained simultaneously. Hwang *et al.* (2005) show that the LPT rule has a tight worst case error bound of $\frac{1}{2}\left\lceil \frac{m}{m-l} \right\rceil$ where l is the number of machines maintained simultaneously, between 1 and m-1.

Liao *et al.* (2005) study the P2|nr-a, single-1, fix|C_{max} and P2|r-a, single-1, fix|C_{max} problems. They partition these problems into four classes depending on the starting and ending times of the maintenance tasks and the total processing time of the jobs. The authors propose optimizing algorithms that have exponential time complexities, implement them on several problem sets and observe that they are quite efficient in solving large-sized problem instances.

Gharbi and Haouari (2005) address the special case of the Pm|nr-a, single, fix, r_j, d_j| C_{max} problem where the maintenance tasks are only at the beginning of the planning horizon. The authors propose a branch and bound algorithm and show that the algorithm can solve instances with up to 700 jobs and 20 machines in a reasonable time.

d) Due Date Related Performance Criteria

Chen (2006b) addresses special cases of the Pm|r-a, multi, nfix-P|$\sum T_j$ and Pm|nr-a, multi, nfix-P|$\sum T_j$ problems where there is a lower bound on the period between two consecutive maintenance tasks. He develops two different mixed binary integer programming models and discusses their relative performance.

3.2 Multi Stage Problems

All studies related with multi stage -flow shop, job shop and open shop- problems with availability constraints consider makespan as the performance measure and variety of those studies is mainly due to the different patterns of the machine unavailability periods, i.e., maintenance activities.

3.2.1 Flow Shop Problems

Lee (1997) is the first researcher who considers availability constraint in a flow shop machine environment. He studies two-machine flow shop makespan problem where one of the machines is always available, i.e., maintenance task is either on the first or on the second machine. The author provides the NP-hardness

results, proposes pseudo-polynomial dynamic programming algorithms and heuristic algorithms having worst case error bounds for both resumable and non-resumable jobs. Cheng and Wang (2000) study a special case of Lee's (1997) problem where the maintenance task is on the first machine and the jobs are resumable. They show that the error bound of the heuristic proposed by Lee (1997) is tight and provide a new heuristic with a smaller worst case error bound. Breit (2004) studies the other special case where the maintenance task is on the second machine and proposes an approximation algorithm whose worst case error bound is smaller than that of the Lee's (1997) heuristic. In his further research, Breit (2006) proposes a polynomial time approximation scheme that can be used to handle both cases.

Lee (1999) extends his former study (Lee, 1997) to the semi-resumable jobs where semi-resumability is a ratio that increases the remaining processing time of a job, but not an additional set up time defined in the other studies. He also considers the case with maintenance tasks on both machines.

Cheng and Wang (1999) study the two-machine flow shop makespan problem where the jobs are semi-resumable and there is one maintenance task on each machine. The maintenance task of one machine starts immediately after that of the other machine ends. The authors propose a heuristic algorithm and provide a worst case error bound for the non-resumable case.

Espinouse *et al.* (1999) address the makespan problem in the two-machine no-wait flow shop with limited machine availability and non-resumable jobs. They provide NP-hardness results for one and several maintenance tasks on one of the machines. Moreover, they propose a polynomial time heuristic algorithm with error bound analysis for one unavailability period case. Wang and Cheng (2001) and Cheng and Liu (2003a, 2003b) provide improved heuristics with smaller worst case error bounds. Cheng and Liu (2003a, 2003b) also consider the case where there are overlapping maintenance tasks on both machines.

Kubzin and Strusevich (2004) also consider the makespan problem in two-machine no-wait flow shops where there is one maintenance task on one of the machines. The difference of their study from the previous research is that they consider all resumable, semi-resumable and non-resumable jobs cases. They

provide complexity results about the problem and propose approximation algorithms with worst case error bounds.

Blazewicz *et al.* (2001) also consider the makespan problem in the two-machine flow shop environment where the jobs are resumable and there are more than one maintenance task on both machines. They propose constructive and local search heuristic algorithms and compare their performances with the optimal solution for the small size problems and with their lower bounds for the large size problems. Their computational results reveal that their heuristic algorithms perform well in solving large-sized problem instances. For this problem, Kubiak *et al.* (2002) develop a branch and bound algorithm that uses a number of dominance rules and Blazewicz *et al.*'s (2001) lower bound.

Aggoune (2004) is the first researcher who considers makespan problem in the *m*-machine flow shop where the jobs are non-resumable and there are several maintenance tasks on each machine. He considers both fixed and non-fixed starting times for maintenance tasks. He proposes tabu search and genetic algorithms and tests their performances.

Ng and Kovalyov (2004) consider the makespan problem in a two-machine flow shop environment where the jobs are resumable and there is a maintenance task on only one of the machines, either first or second. They show that these two cases are equivalent to the partition type problems and propose a fully polynomial time approximation scheme for the latter problem with a finite worst case error bound.

Allaoui and Artiba (2006) investigate the makespan problem in a two-stage hybrid flow shop environment with only one machine in the first stage and *m* machines in the second stage. They assume at most one maintenance task on each machine and non-resumable jobs. The authors present complexity results and provide worst case error bounds for some heuristic algorithms in the literature. They propose a branch and bound algorithm without any computational study.

Allaoui *et al.* (2006) study the makespan problem in a two-machine flow shop environment where there is a maintenance task on the second machine, which is also addressed by Lee (1997). The authors propose dynamic programming algorithms for both resumable and non-resumable jobs. They examine the

performance of modified Johnson's algorithm, discuss conditions under which this heuristic algorithm finds an optimal solution and settle its worst case error bound.

Aggoune and Porttman (2006) address the makespan problem with non-resumable jobs in an *m*-machine flow shop environment where there are several pre-planned maintenance activities. They propose a polynomial time heuristic algorithm and report its satisfactory performance under different problem setting.

3.2.2 Job Shop Problems

Mauguiere *et al.* (2005) address the makespan problem in an *m*-machine job-shop environment where there are several maintenance activities, with both crossable and non-crossable periods, i.e., periods allowing and not allowing the jobs to be resumed, on the machine. They consider both resumable and non-resumable jobs that have release times and latency durations. They propose a branch and bound algorithm and test its performance under different problem settings.

3.2.3 Open Shop Problems

Breit *et al.* (2001) consider the makespan problem in a two-machine open shop environment where the jobs are resumable and one of the machines is maintained once. They show that the problem is NP hard and propose an approximation algorithm with a worst case error bound of $\frac{1}{3}$. Kubzin *et al.* (2006) also study the makespan problem in a two-machine open shop environment and provide polynomial-time approximation schemes for two special cases. In the first case, there is one maintenance task on each machine, and in the second case there are several maintenance tasks, but only on one of the machines only. Breit *et al.* (2003) consider the problem where the jobs are non-resumable. They provide approximation algorithms for one maintenance task on each machine and on the second machine only cases with worst case error bounds of 1 and $\frac{1}{3}$, respectively.

In this section, 51 journal articles appearing in 19 different journals are reviewed. The number of articles published each year, and the number of articles appearing in each journal together with the corresponding publication years are listed in Tables 3.1 and 3.2. As can be observed from Table 3.1, the number of articles considering machine scheduling problems with preventive maintenance activities/unavailability periods has increased significantly in the last decade due to increased recognition and importance of the problem.

Table 3.1 Number of Papers Published Each Year

Year	1989	1990	1991	1992	1993	1994	1995	1996	1997
Number	1	-	1	1	1	1	-	2	2

Year	1998	1999	2000	2001	2002	2003	2004	2005	2006
Number	3	6	4	3	1	5	5	7	8

Table 3.2 Number of Papers Published in Each Journal

Journal	**Number**	**Years**
Acta Informatica	3	1989, 1992, 1998
Annals of Operations Research	2	1997, 2005
Computers & Industrial Engineering	1	1999
Computers & Mathematics with Applications	1	1998
Computers & Operations Research	4	2003, 2006(3)
Discrete Applied Mathematics	7	1991, 1993, 1998, 1999, 2000, 2005(2)
Engineering Optimization	1	2006
European Journal of Operational Research	7	1999, 2000, 2002, 2004(2), 2005(2)
Information Processing Letters	4	1999, 2001, 2003, 2004
International Journal of Production Economics	2	2006(2)
Journal of Global Optimization	1	1996
Journal of the Operational Research Society	3	1999, 2005, 2006
Journal of Scheduling	1	2005
Mathematical and Computer Modelling	1	1994
Mathematical Methods of Operations Research	1	2003
Naval Research Logistics	6	1999, 2000, 2003, 2004(2), 2006
Omega	1	2001
Operations Research Letters	4	1997, 2000, 2001, 2003
Production Planning & Control	1	1996

The number of articles related with each machine environment – performance criterion combination is given in Table 3.3. Note that the total number of articles given in Table 3.3 is greater than 51, as some of the studies consider more than one problem type according to our classification scheme. It should also be noted that, the three review papers are included in the tables given in Tables 3.1 and 3.2 but not in that of Table 3.3. It is seen from Table 3.3 that most of the single machine studies consider the total flow time objective and most of the parallel machine studies consider the makespan objective. It is also observed that all studies related with multi stage environments address the makespan minimization problem.

In this study, we consider the 1|nr-a, multi, fix|$\sum C_j$ and 1|nr-a, multi, fix-P|$\sum C_j$ problems. The most closely related study to ours is due to Chen (2006c). He considers the 1|nr-a, multi, fix-P|$\sum C_j$ problem, unlike ours, with minimum number of batches constraint.

Table 3.3 Number of Papers Related with Each Machine Environment - Performance Criterion Combination

Machine Environment		**Performance Criterion**	**Number**	**Years**
Single Stage	Single Machine	Total Flow Time	9	1989, 1992, 1996, 1999, 2003, 2004, 2005, 2006(2)
		Total Weighted Flow Time	4	1996, 1999, 2005, 2006
		Makespan	2	1996, 2005
		Due Date Related Performance Criterion	4	1996, 1999, 2003, 2006
	Parallel Machine	Total Flow Time	2	1993, 1994
		Total Weighted Flow Time	2	1996, 2000
		Makespan	10	1991, 1996(2), 1998(2), 1999, 2000, 2005(3)
		Due Date Related Performance Criterion	1	2006
Multi Stage	Flow Shop	Makespan	18	1997, 1999(3), 2000, 2001(2), 2002, 2003(2), 2004(4), 2006(4)
	Job Shop	Makespan	1	2005
	Open Shop	Makespan	3	2001, 2003, 2006

CHAPTER 4

OUR SOLUTION APPROACH

There are few studies considering the 1|nr-a, multi, fix|$\sum C_j$ and 1|nr-a, multi, fix-P|$\sum C_j$. Both problems are NP-hard and the associated studies mainly propose heuristic algorithms and their theoretical (worst case bounds) and empirical performances.

In this chapter, we present a branch and bound algorithm to solve the 1|nr-a, multi, fix|$\sum C_j$ problem. As discussed, the 1|nr-a, multi, fix-P|$\sum C_j$ problem is a special case of the 1|nr-a, multi, fix|$\sum C_j$ problem, so the algorithms derived for the latter problem can be used to solve the former problem. In some situations, the special cases may be solved more efficiently due to their properties brought by their special natures. Throughout this chapter, we consider the most general problem 1|nr-a, multi, fix|$\sum C_j$ and state the implementation differences of the periodic case, when necessary.

We now give some definitions and notations used throughout the remaining part of this study in Section 4.1, and afterwards present the properties of an optimal solution, upper and lower bounds on the optimal value of the total flow time in Sections 4.2, 4.3 and 4.4, respectively. Finally, we define our branch and bound algorithm in Section 4.5.

4.1. Some Definitions and Notations Used

Let *F(S)* denote the total flow time of a feasible schedule S which has *q(S)* batches and batch *k* has $n_k(S)$ jobs. *F(S)* is an upper bound on the optimal value of the total flow time and is composed of two parts: the processing times (*J(S)*) and the starting times and durations of maintenance tasks (*M(S)*). For the sake of simplicity, we drop a reference to schedule S, e.g., we let F denote the total flow time of S in place of $F(S)$.

Let $[jk]$ be the j^{th} job in batch k and $C_{[jk]}$ be the completion time of job [*jk*]. $C_{[jk]}$ can be expressed as

$$C_{[jk]} = s_{k-1} + t_{k-1} + \sum_{i=1}^{j} p_{[ik]} \tag{4.1}$$

Let T_k be the length of the interval between k^{th} and $(k-1)^{th}$ maintenance tasks. Then $T_k = s_k - s_{k-1} - t_{k-1}$ and $C_{[jk]}$ becomes

$$C_{[jk]} = \sum_{r=1}^{k-1} (T_r + t_r) + \sum_{i=1}^{j} p_{[ik]} \tag{4.2}$$

Summing the completion times of all jobs in batch *k*, we obtain the total completion time of batch *k* as

$$\sum_{j=1}^{n_k} C_{[jk]} = n_k \sum_{r=1}^{k-1} (T_r + t_r) + \sum_{j=1}^{n_k} \sum_{i=1}^{j} p_{[ik]} \tag{4.3}$$

Note that, processing time of j^{th} job in batch k, $p_{[jk]}$, appears in all $C_{[ik]}$ such that $j \le i \le n_k$. Thus, the coefficient of $p_{[jk]}$ in the total completion time expression is $n_k - j + 1$ and the completion time of batch k can be rewritten as:

$$\sum_{j=1}^{n_k} C_{[jk]} = n_k \sum_{r=1}^{k-1} (T_r + t_r) + \sum_{j=1}^{n_k} (n_k - j + 1) p_{[jk]} \tag{4.4}$$

Summing the total completion times of all batches, we obtain the total flow time of schedule *S* as:

$$F = \sum_{k=2}^{q} \sum_{r=1}^{k-1} n_k (T_r + t_r) + \sum_{k=1}^{q} \sum_{j=1}^{n_k} (n_k - j + 1) p_{[jk]} \tag{4.5}$$

The first part of the *F(S)* given in equation (4.5) is the contribution of the starting times and durations of the maintenance tasks to the total flow time.

$$M = \sum_{k=2}^{q} \sum_{r=1}^{k-1} n_k (T_r + t_r) \tag{4.6}$$

The total contribution of the processing times to the flow time is the second part of equation (4.5).

$$J = \sum_{k=1}^{q} \sum_{j=1}^{n_k} (n_k - j + 1) p_{[jk]} \tag{4.7}$$

The sum of processing times of the jobs in batch k is the used portion of the processing capacity of the corresponding interval and named as content of batch k, TP_k.

$$TP_k = \sum_{j=1}^{n_k} p_{[jk]} \tag{4.8}$$

Slack time, or idle time of a batch k, I_k, is the unused processing capacity of the corresponding interval.

$$I_k = T_k - \sum_{j=1}^{n_k} p_{[jk]} = T_k - TP_k \tag{4.9}$$

4.2. Properties of an Optimal Solution

Property 1. The jobs within each batch are ordered in nondecreasing order of their processing times, i.e., SPT order, in all optimal solutions. (see Section 2.3)

Property 2. If $I_k > p_j$, then job j is sequenced in batch r where $r \le k$, in all optimal solutions.

Proof. If job j is sequenced in a later batch r', such that $r' > k$, then the total completion time can be reduced by removing that job from batch r' and inserting it to batch k. Such a replacement is feasible as the slack time of batch k, i.e., I_k, is no smaller than p_j. The start times, thus the completion times, of the jobs sequenced later than job j in batch r' will decrease by p_j units, the start time of job j will decrease as it is sequenced in an earlier batch, and the start times of other jobs will not be effected from this interchange. Hence, a schedule in which job j is sequenced in a later batch than batch k cannot be optimal.

It should be pointed that, a similar property for non-fixed periodic case is proposed by Qi *et al.* (1999) and Aktürk *et al.* (2003, 2004)).

Property 3. In the optimal schedule, if job job j is placed at a^{th} position from last in batch k and i is sequenced at b^{th} position from last in batch r, then $p_i > p_j$ implies that either $a \geq b$ or $I_k < p_i - p_j$.

Proof. Let S be a feasible schedule in which $p_i > p_j$, $a < b$ and $I_k \geq p_i - p_j$ i.e., S does not satisfy the condition of the property. Note that, the coefficients of p_j and p_i are a and b, respectively, due to their position within the batch. Let S' be the schedule obtained by interchanging jobs i and j while keeping the sequences of other jobs fixed. Such an interchange is feasible as $I_k \geq p_i - p_j$, i.e., job i fits to batch k after job j is removed, and $p_j < p_i$, i.e., job j fits to batch r after job i is removed. With this interchange, total completion time is increased by $a*(p_i - p_j)$ and decreased by $b*(p_i - p_j)$. So, total completion time of S' is:

$$\begin{aligned} F(S') &= F + a*(p_i - p_j) - b*(p_i - p_j) \\ &= F + (a-b)(p_i - p_j) \end{aligned}$$

Note that $(a-b)(p_i - p_j)$ is negative since $p_i > p_j$ and $a < b$ are positive. So, $F(S') < F$, i.e., there is a schedule with smaller total flow time value than that of S. Hence, a schedule that does not satisfy the condition of the property cannot be optimal.

Property 4. In any optimal solution, if the number of jobs in batch k is smaller than the number of jobs in batch r, i.e., $n_k < n_r$, where $k < r$ then either $TP_r > T_k$ or $TP_k > T_r$.

Proof. Let S be a feasible schedule in which $n_k < n_r$, $TP_r < T_k$ and $TP_k < T_r$, i.e., the condition of the property does not hold. If S' is the schedule obtained by interchanging all jobs in batch k with the jobs in batch r of S, then

$$\begin{aligned} F(S') &= F - n_r[(s_{r-1} + t_{r-1}) - (s_{k-1} + t_{k-1})] + n_k[(s_{r-1} + t_{r-1}) - (s_{k-1} + t_{k-1})] \\ &= F - (n_r - n_k)[(s_{r-1} + t_{r-1}) - (s_{k-1} + t_{k-1})] \end{aligned}$$

As $(n_r - n_k)$ and $[(s_{r-1} + t_{r-1}) - (s_{k-1} + t_{k-1})]$ greater than zero, $(n_r - n_k)[(s_{r-1} + t_{r-1}) - (s_{k-1} + t_{k-1})]$ is positive, thus $F(S') < F$. Hence, S, i.e., a schedule that does not satisfy the conditions of the property cannot be optimal.

Property 5. (Chen, 2006c) In any optimal schedule, if $T_k = T$ for all k, $n_k \geq n_{k+1}$.

Proof. This property is just a direct implication of Property 4 to the periodic maintenance case and as interchanging any consecutive batches is feasible in periodic maintenance case, i.e., $TP_{k+1} \leq T_k$ and $TP_k \leq T_{k+1}$ for all k, $n_k \geq n_{k+1}$ for all k.

4.3 Upper Bounds

Remember that SPT minimizes total flow time when there is no maintenance task. Hence, one should expect satisfactory performance from SPT, when T_k s are long and t_k s are short. In such a case, there will be smaller number of batches and the completion times will be shorter due to small t_k and few batches. This will imply smaller contribution of M to F.

On the other hand, when T_k s are short and t_k s are long, the M value will be significant in F. This is due to having higher number of batches because of short T_k s, and higher completion times due to long t_k s and more batches. In such an environment, the SPT may not perform satisfactory as it may keep high idle times due to forcing the jobs of the same magnitude in the same batch.

To summarize, en route to reducing J, one should favor SPT, however en route to reducing M one should favor a schedule with minimum number of batches.

Total completion time value of any feasible schedule is an upper bound on the optimal value of total completion times and in this section, we present the heuristic procedures that we employ to derive a global upper bound by generating an initial feasible schedule.

4.3.1 Improved SPT Heuristic (ISPT)

ISPT constructs a feasible sequence by using SPT order, and afterwards improves this sequence by using pairwise exchanges of the jobs in different batches. Remembering that the jobs are indexed in SPT order, below is the stepwise description of the ISPT heuristic:

Construction

Step 0. $j=1, k=1$

Step 1. If job j fits to batch k, i.e., slack time of batch k is not less than p_j assign job j to batch k.

If it does not fit to batch *k*, go to Step 3.

Step 2. If *j=n* stop, all jobs are sequenced and the number of batches $q=k$.

Let $j=j+1$ and go to Step 1.

Step 3. Let $k=k+1$ and go to Step 1.

Improvement step takes the resulting q batches and improves the solution by assigning a larger processing time to earlier batches with the hope of reducing the idle times of the earlier batches without increasing its cardinality. After we get the initial schedule, starting with batch 1, we compare each batch with the succeeding batches. We allow pairwise interchanges between the shorter job of batch k and longer job of batch r, $r>k$. When no further interchanges can be made between batch k and the succeeding batches, we proceed to batch $k+1$ and compare it with latter batches. We terminate when we reach the last batch, i.e., batch q. Below is the stepwise description of improvement step.

Improvement

S is the schedule obtained by construction phase.

Step 0. Let $k=1$.

Step 1. If $k=q$, stop. *S* is the schedule obtained by ISPT.

Step 2. If $I_k=0$, let $k=k+1$ and go to Step 1.

Step 3. Let $j=$ the last job of batch $k=n_k$.

Step 4. Let $r = k+1$.

If $r > q$, let $k = k+1$ and go to Step 1.

Step 5. If $r \le q$, then let $i = 1$.

Else, go to Step 8.

Step 6. If $[jk]$ and $[ir]$ can be interchanged feasibly, $p_{[jk]} < p_{[ir]}$ and flow time does not increase due to this interchange, then swap these two jobs, update S and the associated idle times: $I_k = I_k + p_{[ir]} - p_{[jk]}$, $I_r = I_r + p_{[jk]} - p_{[ir]}$.

If swapping $[jk]$ and $[ir]$ is not feasible and processing times of all jobs after $[ir]$ is greater than $p_{[ir]}$, go to Step 8.

If $I_k = 0$, let $k = k+1$ and go to Step 1.

Step 7. Let $i = i+1$.

If $i > n_r$, let $r = r+1$ and go to Step 5.

Else, go to Step 6.

Step 8. Let $j = j-1$.

If $j = 0$, let $k = k+1$ and go to Step 1.

Else, let $r = k+1$ and go to Step 5.

4.3.2 Modified SPT Heuristic (MSPT)

The MSPT heuristic is a constructive type which follows SPT while forming the batches. When a batch has to be closed due to the absence of a fittable job, i.e., the idle time is less than the processing time of the next job of SPT, we check whether it is possible to reduce the idle time with a pairwise exchange of the jobs in the batch and unscheduled jobs. Accordingly, if there exists an unassigned job that can fit to the batch when an assigned job is removed, the composition of the batch is changed by removing the shorter job and assigning the longer job. The aim is to reduce the number of batches and the completion times of the unscheduled jobs.

Below is the stepwise description of the heuristic that we call MSPT.

Let S_k = the ordered set of jobs in batch *k*.

Step 0. Let $k=1,\ S_k=\varnothing,\ US=\{1,...,n\}$.

Step 1. Assign the jobs in US to batch k until no job fits to batch k or all jobs are already assigned.

Update S_k and US after each job assignment.

If $US=\varnothing$, then stop. $S=\bigcup_{k=1}^{q(S)} S_k$ is the feasible schedule obtained by MSPT.

Step 2. If $I_k=T_k-\sum_{j=1}^{n_k} p_{[jk]}=0$, i.e., the batch is full, let $k=k+1$ and go to Step 1.

If $p_{[n_k k]}+I_k<\min_{i\in US}\{p_i\}$, i.e., there is no feasible exchange of jobs in S_k and US, let $k=k+1,\ S_k=\varnothing$ and go to Step 1.

Else, i.e., it is feasible to replace the last assigned job in batch k with the shortest unscheduled job, let $j=1$.

Step 3. If $j>n_k$, go to Step 6.

Step 4. If $p_{[jk]}+I_k<\min_{i\in US}\{p_i \mid p_i>p_{[jk]}\}$, i.e., there is not a feasible exchange of job $[jk]$ with a longer unscheduled job, let $j=j+1$ and go to Step 3.

Exchange j^{th} job of batch k with the longest unscheduled job that can feasibly be swapped, i.e., job i such that $p_i=\max_{h\in US}\{p_h \mid p_h\le p_{[jk]}+I_k\}$.

Update S_k, I_k and *US*.

Step 5. If $I_k>0$, let $j=j+1$ and go to Step 3.

Step 6. Reorder the jobs in batch *k* in the SPT order.

Let $k=k+1$ and go to Step 1.

4.3.3 Numerical Example

We illustrate our upper bounding procedures by using the following example with $n=10$ jobs and periodic maintenance tasks with $T=20$ and $t=4$. The processing times of the jobs are given below:

j	1	2	3	4	5	6	7	8	9	10
p_j	2	3	3	4	5	6	7	9	10	10

Note that the jobs are indexed by SPT order.

ISPT

Construction

Assigning the jobs to the batches in nondecreasing order of their processing times, the initial schedule S having $q = 4$ batches is obtained. S is shown in Figure 4.1.

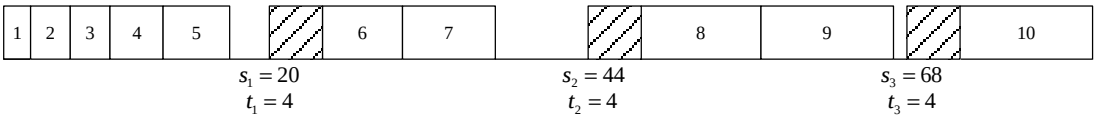


Figure 4.1 Initial Schedule Obtained by the Construction Phase of ISPT Heuristic

The total flow time of S is

$$F = \sum_{j=1}^{10} C_j = 2+5+8+12+17+30+37+57+67+82 = 317 .$$

Improvement

The details of the improvement phase can be seen in Appendix 4.1. Here, we only illustrate the schedules after each completion of all pairwise interchanges between the jobs in a batch and the jobs in the succeeding batches.

After the pairwise interchanges between the jobs in the first batch and the succeeding batches, S becomes

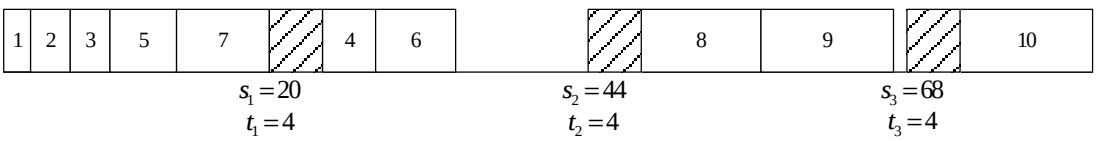


Figure 4.2 The Schedule After the Pairwise Interchanges of Jobs in Batch 1 and the Succeeding Ones

After the pairwise interchanges between the jobs in the second batch and the succeeding batches, S becomes

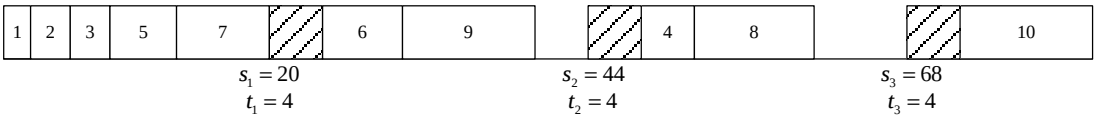


Figure 4.3 The Schedule After the Pairwise Interchanges of Jobs in Batch 2 and the Succeeding Ones

Finally, after the pairwise interchanges between the jobs in the third batch and the last batch, S becomes

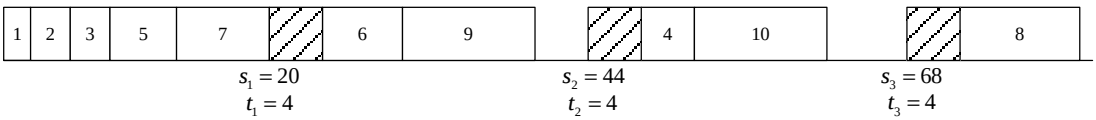


Figure 4.4 The Schedule After the Pairwise Interchanges of Jobs in Batch 3 and the Succeeding Ones

S is the schedule obtained by ISPT and the corresponding total flow time is

$$F = \sum_{j=1}^{10} C_j = 2+5+8+13+20+30+40+52+62+81 = 313 .$$

MSPT

The details of the MSPT heuristic can be seen in Appendix 4.2. Here, we only illustrate the schedules after each batch is constructed and modified.

Batch 1:

Initially, jobs 1,2,3,4,5 are put to batch 1.

Job 2 is interchanged with Job 6.

As there is no more possible exchanges, the job sequence corresponding to batch 1 becomes $S_1 = \{1,3,4,5,6\}$ with zero idle time, and the set of unscheduled jobs is $US = \{2,7,8,9,10\}$.

Batch 2:

Jobs 2,7,8 are put to batch 2.

Job 8 is interchanged with Job 10.

As there is no more possible exchanges, the job sequence corresponding to batch 2 becomes $S_2 = \{2,7,10\}$ with zero idle time, and the set of unscheduled jobs is $US = \{8,9\}$.

Batch 3:

All remaining unscheduled jobs, i.e., jobs 8 and 9, are put to batch 3.

As there is no more unscheduled jobs, i.e., $US = \varnothing$, the final schedule obtained by ISPT heuristic is $S = S_1 \cup S_2 \cup S_3 = \{1,3,4,5,6,2,7,10,8,9\}$, which is shown in Figure 4.5. The total flow time value of S is $F = \sum_{j=1}^{10} C_j = 2+5+9+14+20+27+34+44+57+67 = 279$.

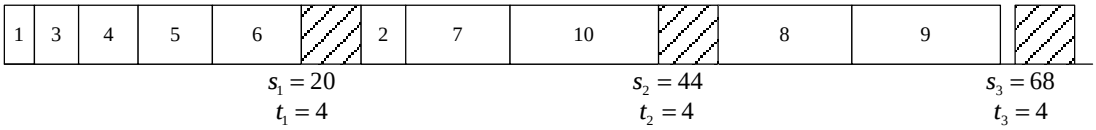


Figure 4.5 Initial Schedule Obtained by MSPT Heuristic

4.4 Lower Bounds

Lower bounding is the key mechanism of a branch and bound algorithm and in this section we propose two lower bounds on the optimal value of the total flow time.

4.4.1 Lower Bound 1

The 1|r-a, multi, fix|$\sum C_j$ problem is a relaxation of our problem where the jobs are allowed to be resumable. The optimal solution of the relaxed problem is obtained by the Preemptive SPT (PSPT) rule (Lee, 1996). As any relaxation of a problem yields a lower bound, the total flow time value of the PSPT schedule is a lower bound on the optimal total flow time value of the 1|nr-a, multi, fix|$\sum C_j$ problem.

Remember that the jobs are indexed in SPT order, C_j and t_k denote the completion time of job j and the duration of maintenance k, respectively. We let LB_1 denote the lower bound value obtained by the PSPT rule.

Below is the stepwise description of the lower bound procedure 1, LB_1.

Step 0. Let $j=1, k=1, C_0=0, LB_1=0$

Step 1. If job j does not fit to batch k, let $C_j = C_{j-1} + t_k + p_j$ and $LB_1 = LB_1 + C_j$, and go to Step 3.

If job j fits to batch k, i.e., slack time of batch k is not less than p_j, $C_j = C_{j-1} + p_j$ and $LB_1 = LB_1 + C_j$.

Step 2. If $j=n$ stop, all jobs are sequenced.

Else, let $j = j+1$ and go to Step 1.

Step 3. Let $k = k+1$ and go to Step 1.

If the PSPT sequence includes no job that is preempted by a maintenance task, i.e. each maintenance start time corresponds to a completion time of a job, then it is feasible, hence optimal solution of the 1|nr-a, multi, fix|$\sum C_j$ problem.

4.4.2 Lower Bound 2

Our second lower bound, LB_2, finds upper bounds on the number of jobs that can be processed in each batch and then assign the jobs to the batches optimally using these bounds. If the resulting solution is feasible, when the total processing times assigned to all batches are no bigger than the durations of the maintenance periods, then it is optimal to our problem, otherwise the cost of the solution is a lower bound.

We let sub_k denote the upper bound on the number of jobs that can be assigned in batch k and $csub_k$ denote the upper bound on the cumulative number of jobs that can be assigned to batches 1 through k. Thus, sub_k

satisfies $\sum_{j=1}^{sub_k} p_j \le T_k$ and $\sum_{j=1}^{sub_k+1} p_j > T_k$, and $csub_k$ satisfies $\sum_{j=1}^{csub_k} p_j \le \sum_{r=1}^{k} T_r$ and $\sum_{j=1}^{csub_k+1} p_j > \sum_{r=1}^{k} T_r$. Note that, $csub_k$ is equal to the number of jobs completed in batch k in the PSPT schedule as cumulative capacity consideration corresponds to relaxing the non-resumability of the jobs.

We now discuss, how we find an upper bound on the number of jobs assigned to batch k, nub_k, given the upper bound on the number of jobs in batches 1 through $k-1$.

Note that $nub_1 = sub_1$. For general k, we use

$$nub_k = \min\{sub_k, csub_k - \sum_{r=1}^{k-1} nub_r\} \tag{4.10}$$

Clearly sub_k is an upper bound on the number of jobs that can be processed in batch k. Moreover, we include the second term $csub_k - \sum_{r=1}^{k-1} nub_r$ which is an upper bound on the number of jobs to be assigned to batch k, given an upper bound of $\sum_{r=1}^{k-1} nub_r$ jobs in batches 1 through $k-1$.

We now determine an upper bound on the number of batches used, given that nub_k jobs are processed in batch k. Note that the resulting upper bound is h where satisfies $\sum_{k=1}^{h-1} nub_k < n$ and $\sum_{k=1}^{h} nub_k \ge n$.

Assuming that there are h batches and nub_k jobs in batch k, we can write the total flow time expression as follows

$$LB_2 = \sum_{k=2}^{h} \sum_{r=1}^{k-1} nub_k (T_r + t_r) + \sum_{k=1}^{h} \sum_{j=1}^{nub_k} (nub_k - j + 1) p_{[jk]} \tag{4.11}$$

where $[jk]$ is the j^{th} sequenced job of batch k.

The expression in (4.11) is lower bound on the total flow time provided that the second term is minimized.

The second term is associated with the job sequencing decision, i.e., finding $[jk]$s. $\sum_{k=1}^{h}\sum_{j=1}^{nub_k}(nub_k - j + 1)p_{[jk]}$ is a product sum of two sets: weight and processing time sets. The decision is to assign each element in weight set to one element in processing time set, so as to get a one to one matching.

The elements in the weight set can be written as 1,2,…, nub_1,1,2,…, nub_2,…,1,2,… nub_h. The sorted set is then {1,1,..,1,2,2,…2,…, $w_{\max}$, $w_{\max}$,…} where $w_{\max} = \max_{1\le k\le h}\{nub_k\}$. Assume the number of times value "w" appears in the set is l_w.

An optimal solution to our matching problem having a product sum structure is max to min matching. Max to min matching assigns i^{th} smallest weight to i^{th} largest processing time. Accordingly, the first $l_1 = h$ largest processing times are assigned to weight "1" and the contribution to the second term of (4.11) is $\sum_{i=1}^{l_1} p_{[n-i+1]}$ where $[i]$ is the job with the i^{th} smallest processing time. Similarly, the next l_2 largest processing times are assigned to weight "2" with contribution of $\sum_{i=l_1+1}^{l_1+l_2} 2p_{[n-i+1]}$. In general, weight "$w$" is assigned to $cl_{w-1}+1$ through cl_w largest processing times where $cl_w = \sum_{i=1}^{w} l_i$. The total contribution to second term of (4.11) is then

$$\sum_{w=1}^{w_{\max}}\sum_{i=cl_{w-1}+1}^{cl_w} wp_{[n-i+1]} \tag{4.12}$$

We now substitute the above term in place of $\sum_{k=1}^{h}\sum_{j=1}^{nub_k}(nub_k - j + 1)p_{[jk]}$ in (4.11) and get lower bound 2, LB_2.

$$LB_2 = \sum_{k=2}^{h}\sum_{r=1}^{k-1} nub_k(T_r + t_r) + \sum_{w=1}^{w_{\max}}\sum_{i=cl_{w-1}+1}^{cl_w} wp_{[n-i+1]} \tag{4.13}$$

Note that, if we let $w_{[i]}$ denote the i^{th} element in the sorted set of weights, then the optimal solution of the matching problem can also simply be expressed as

$$\sum_{i=1}^{n} w_{[i]} p_{[n-i+1]} \tag{4.14}$$

Then, we have LB_2 as

$$LB_2 = \sum_{k=2}^{h} \sum_{r=1}^{k-1} nub_k (T_r + t_r) + \sum_{i=1}^{n} w_{[i]} p_{[n-i+1]} \tag{4.15}$$

Note that once the matching problem produces a feasible solution, it is optimal to the problem. A solution is feasible if nub_k assigned jobs fit to batch k. There are many alternative optimal solutions to the matching problem as there are many identical weights. The number of alternative solutions is $\prod_{w=1}^{w_{\max}} l_w$ and it is not computationally efficient to all en route to obtaining a feasible, hence an optimal solution. In place of generating all these solutions, we propose the following procedure with the hope of finding a feasible solution.

Let w_j be the weight assigned to job j in the matching solution. Starting from the longest job, we assign job j to batch accommodating weight w_j and having largest idle time.

Our aim here is to assign longer job to the more available batch, hence balance the processing load among the batches. Such a balancing would increase the chance of getting a feasible solution.

4.4.3 Numerical Example

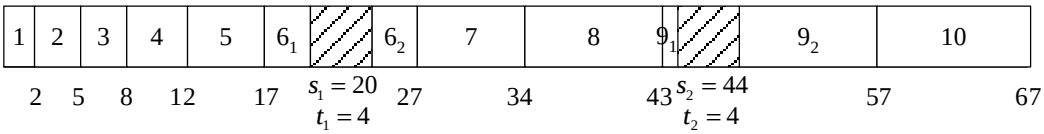
We now illustrate our lower bounding procedures by using the example given in Section 4.3. Remember that the example includes $n = 10$ jobs and periodic maintenance tasks with $T = 20$ and $t = 4$, and the processing times of the jobs are:

j	1	2	3	4	5	6	7	8	9	10
p_j	2	3	3	4	5	6	7	9	10	10

Lower Bound 1

Total processing time of the first five jobs is 17, so they are completely assigned to the first batch. The remaining capacity of the first batch is 3 and is

smaller than the processing time of job 6, so job 6 is preempted by the first maintenance task and resumed after it. After job 6 is completed, job 7 and 8 are assigned to the second batch and job 9 is preempted by the second maintenance task and resumed after it. Third batch has still enough capacity to include the last job completely, so job 10 is assigned to the third batch and the PSPT schedule is obtained. Total flow time of the sequence is $F = \sum_{j=1}^{10} C_j = 2 + 5 + 8 + 12 + 17 + 27 + 34 + 43 + 57 + 67 = 272$ and the schedule is illustrated by the Gantt chart below.



Lower Bound 2

Determining sub_k values

As the maintenances are periodic in the example, sub_k values are the same. It can be observed from the PSPT schedule built for lower bound 1 that $sub_k = 5$, i.e., the first five jobs can entirely be processed in a batch and the sixth job does not fit to the remaining capacity. Note that $\sum_{j=1}^{5} p_j \le 20$ and $\sum_{j=1}^{6} p_j > 20$.

Determining $csub_k$ values

$csub_1 = sub_1 = 5$

$\sum_{j=1}^{8} p_j \le T_1 + T_2 = 40$ and $\sum_{j=1}^{9} p_j > T_1 + T_2 = 40$. So, $csub_2 = 8$.

$n = 10$ and $\sum_{j=1}^{10} p_j \le T_1 + T_2 + T_3 = 60$. So, $csub_3 = 10$.

As there are 10 jobs and the sum of their processing times is no more than the total capacity of the first 3 batches, $csub_k = 10$ for all $k \ge 3$.

Determining nub_k values

$$nub_1 = sub_1 = 5$$

$$nub_2 = \min\{sub_2, csub_2 - nub_1\} = \min\{5, 8-5\} = 3$$

$$nub_3 = \min\{sub_3, csub_3 - (nub_1 + nub_2)\} = \min\{5, 10-(5+3)\} = 2$$

Using these nub_k values, the contribution of the maintenance tasks to the total flow time is:

$$nub_1(0) + nub_2(T_1 + t_1) + nub_3((T_1 + t_1) + (T_2 + t_2))$$

$$= 5(0) + 3(20+4) + 2((20+4) + (20+4)) = 168$$

Solving the Matching Problem

The weight set corresponding to the determined nub_k values is {1,2,3,4,5,1,2,3,1,2}.

The sorted weighted set is {1,1,1,2,2,2,3,3,4,5} and the set of processing times in nonincreasing order is {10,10,9,7,6,5,4,3,3,2}.

Assigning the larger processing times to the smaller weights, optimal solution to the matching problem is obtained as

1(10)+1(10)+1(9)+2(7)+2(6)+2(5)+3(4)+3(3)+4(3)+5(2)=108.

Determining the Total Flow Time

The total flow time is the sum of the contribution of maintenance tasks and processing times, $LB_2 = 168 + 108 = 276$.

4.5 Branch and Bound Algorithm

The bounding and reduction mechanisms are used in our branch and bound algorithm.

The optimal schedule can be found by dividing a sequence of n jobs into subsequences, where each subsequence corresponds to a batch. Due to Property 2, a batch should never be closed if there is a job that fits in. Hence, the addition of a job to a sequence represents the addition of a job to the current batch if the capacity

of the batch is not violated. If the capacity is violated, the addition represents the creation of the next batch. Hence, to find the optimal solution, it is sufficient to evaluate $n!$ possible sequences. We use a depth first strategy to evaluate these sequences. We favor this strategy due to its relatively low memory requirements.

At level l of the branch and bound tree, the decision about the addition of the l^{th} job of the sequence is made. The complete solution is reached at level n.

To avoid the non-SPT schedules within each batch (see Property 1), we discard nodes corresponding to the addition to the current batch if the index of the unscheduled job is smaller. Hence, if job i is the last assigned job to the current batch, then there can be at most $(n-i)$ additions. Moreover, to avoid duplication of the sequences due to identical processing times, we only consider the jobs with distinct processing times. Hence, we consider only job i, but not job j, if $p_i = p_j$. This follows that there can at most d additions if there are d distinct processing times.

In presenting the implementation of optimality properties and bounding mechanisms, we let s denote the current partial schedule, and $US(s)$ denote the set of jobs that does not appear in s, i.e., set of unsequenced jobs.

We discard nodes that correspond to closing the k^{th} batch if the following conditions hold.

a) There exists an unscheduled job that can fit to the current batch. (Otherwise Property 2 is violated.)

b) There exists an unscheduled job i such that $I_k + p_j - p_i \geq 0$ and $p_i > p_j$ where job j is the last job of the k^{th} batch. (Otherwise the special case of Property 3 where $a=1$ and $r>k$ is violated. Note that, the position of unscheduled job i in batch r is not important as it is no less than a)

Once we fathom s due to Property 3, we proceed to the partial schedule $s \setminus \{j\} \cup \{i'\}$ where job i' is the longest job that can fit to k^{th} batch, once job j is removed. Note that, the partial schedule $s \setminus \{j\} \cup \{i\}$ where $p_i < p_{i'}$ would be eliminated by $s \setminus \{j\} \cup \{i'\}$ due to Property 3.

c) There exists already scheduled batch r $(r < k)$ such that $n_r < n_k$ and batch r and batch k can be feasibly interchanged. (Otherwise Property 4 or Property 5 is violated for non-periodic and periodic cases, respectively.)

d) There exists already scheduled batch r $(r < k)$ such that $n_r = n_k$ and batch r and batch k can be feasibly interchanged and one of the following conditions hold.

- The sum of the indices of the jobs in batch k is smaller than the sum of the indices of the jobs in batch r.
- The sum of the indices of the jobs in batch k is equal to the sum of the indices of the jobs in batch r and the index of the first job in batch k is smaller than that of batch r.

The above conditions are imposed as interchanging batches k and r produces alternative solutions one of which should be avoided for the efficient implementation of the branch and bound algorithm.

We use Property 4, to discard the nodes even when the batch does not have to be closed. In doing so, we calculate a lower bound on the number of jobs that can fit to the remaining positions of batch k. A valid lower bound is nl_k where nl_k satisfies $\sum_{i=1}^{nl_k} p_{[n_{US(s)}-i+1]} \le I_k$ and $\sum_{i=1}^{nl_k+1} p_{[n_{US(s)}-i+1]} > I_k$ where $p_{[n_{US(s)}-i+1]}$ is the processing time of the i^{th} largest unscheduled job and $n_{US(s)}$ is the number of unscheduled jobs at the current node. We fathom the node whenever $n_r < n_k + nl_k$, $T_k \le T_r$ and the jobs in batch r can fit to batch k, i.e., the sum of the processing times in batch r is no more than T_k.

We calculate a lower bound for each node that cannot be eliminated by the optimality properties. We use the following additional notation to discuss the implementation of the lower bounding mechanisms to s.

$j(s)$: the index of the job added last to s

$k(s)$: the current batch in s

We close the current batch if there is no unscheduled job that fits. We then find a lower bound, $LB_1(s)$, by assigning all unscheduled jobs to the subsequent batches, i.e., the batches $k(s)+1$, $k(s)+2...$, according to the PSPT rule.

If there is at least one job in US that fits to batch $k(s)$, i.e., $I_{k(s)} \geq \min_{j \in US}\{p_j\}$, then we find a lower bound by only allowing the jobs j such that $j > j(s)$ to appear in batch $k(s)$. Hence, the resulting lower bound problem can be solved by assuming the jobs j such that $j < j(s)$ become ready at the start time of batch $k(s)+1$. The optimal preemptive solution is to sequence the job having smallest processing time among the available jobs (see Pinedo, 1995). Hence, a job that violates the SPT order will never appear in batch $k(s)$.

We let s_j denote a partial schedule in which job j is appended to s. Through the following property we establish a relation between $LB_1(s_i)$ and $LB_1(s_j)$ for two unscheduled jobs i and j.

Property 6. If $i < j$, i.e., $p_i \leq p_j$, then $LB_1(s_i) \leq LB_1(s_j)$.

Proof. Note that the lower bound problems solved in s_i and s_j are identical except that they differ by ready times. The ready times of the jobs in s_i (s_j) are zero if their processing times are no smaller than p_i (p_j), and the start time of batch $k(s)+1$ if their processing times are smaller than p_i (p_j). As $j > i$, the ready times in s_i are never greater, hence the associated total flow time value, $LB_1(s_i)$, is never greater than $LB_1(s_j)$.

We fathom s if $LB_1(s) \geq UB$ where UB is the best known upper bound.

If there are no preempted jobs in the sequence corresponding to $LB_1(s)$, i.e., it is feasible and $LB_1(s) < UB$, then we update the best upper bound known as $UB = LB_1(s)$ and we fathom s.

Using the result of Property 6, we never consider s_j for all $j > i$ if $LB_1(s_i) \geq UB$.

We implement the lower bounds sequentially. If partial schedule s cannot be fathomed by $LB_1(s)$, we calculate the second lower bound $LB_2(s)$.

We update the upper bounds on the number of jobs in batch $k(s)$ and succeeding batches considering only the unscheduled jobs, $US(s)$.

To determine an upper bound on the number of jobs that can be assigned to the remaining positions of batch $k(s)$, we only consider the jobs having higher indices than $j(s)$ and the remaining capacity of batch $k(s)$, i.e., $I_{k(s)}$. We let $[j]$ denote the j^{th} shortest unscheduled job. Then, $sub_{k(s)}$ satisfies $\sum_{\substack{1\le j\le sub_{k(s)} \\ p_{[j]}\ge p_{j(s)}}} p_{[j]} \le I_{k(s)}$ and $\sum_{\substack{1\le j\le sub_{k(s)}+1 \\ p_{[j]}\ge p_{j(s)}}} p_{[j]} > I_{k(s)}$. For the succeeding batches k such that $k > k(s)$, sub_k satisfies $\sum_{j=1}^{sub_k} p_{[j]} \le T_k$ and $\sum_{j=1}^{sub_k+1} p_{[j]} > T_k$.

For partial schedule s, $csub_k$ is an upper bound on the number of jobs that can be assigned to batches $k(s)$ through k considering the cumulative capacity $I_{k(s)} + \sum_{r=k(s)}^{k} T_r$. Note that, $csub_k$ values are already determined by $LB_1(s)$ since considering the cumulative capacity of batches implies resumability of the unscheduled jobs. Hence, $csub_k$ is the number of jobs completed in batch k in the $LB_1(s)$ sequence.

Note that $nub_{k(s)} = csub_{k(s)} = sub_{k(s)}$ as $k(s)$ is the first batch of consideration. Given sub_k and $csub_k$ values, an upper bound on the number of jobs that can be assigned to succeeding batch k, nub_k, is,

$$nub_k = \min\{sub_k, csub_k - \sum_{r=k(s)}^{k} nub_k\} \tag{4.16}$$

Assume $h(s)$ satisfies $\sum_{k=k(s)}^{h(s)-1} nub_k < n_{US}(s)$ and $\sum_{k=k(s)}^{h(s)} nub_k \ge n_{US}(s)$ where $n_{US}(s)$ is the number of unscheduled jobs at the current node. Note that, $h(s)$ is the number of batches that will be considered in our bound computations.

The completion time of the last job in the partial schedule s , $C_{j(s)}$, is the starting time of the first job to be scheduled in batch $k(s)$, which affects the flow times of $nub_{k(s)}$ jobs. For any succeeding batch k, the flow times of nub_k jobs are affected by the starting time of batch k which is $s_k = \sum_{r=1}^{k-1}(T_r + t_r)$. Then, the contribution of $C_{j(s)}$ and the succeeding maintenance tasks to $LB_2(s)$ is,

$$M(s)_{LB_2} = nub_{k(s)}C_{j(s)} + \sum_{k=k(s)+1}^{h(s)} \sum_{r=1}^{k-1} nub_k(T_r + t_r) \tag{4.17}$$

The contribution of processing times to $LB_2(s)$ is accounted by the following weight formed by the updated nub_k values (see Section 4.4)

$\{1,2,\ldots, nub_{k(s)}, 1,2,\ldots, nub_{k(s)+1}, \ldots, 1,2,\ldots, nub_{h(s)}\}$.

Note that the number of elements in the weight set, i.e., $\sum_{k=k(s)}^{h(s)} nub_k$, is equal to the number of the unscheduled jobs at the current node, i.e., $n_{US(s)}$.

Denoting the maximum element in the weight set by $w_{\max}(s)$, i.e., $w_{\max}(s) = \max_{k(s)\le k\le h(s)}\{nub_k\}$, and sorting the weights, the ordered set of weights becomes

$\{1,1,..,1,2,2,\ldots 2,\ldots, w_{\max}(s)\}$

To minimize the resulting product sum, the smaller weights are matched with the larger processing times as described in Section 4.4. So, if we let $l_{w(s)}$ and $cl_{w(s)}$ denote the number of $w(s)$s in the weight set and $\sum_{i=1}^{w(s)} l_i$, respectively, the minimum product sum of weights and processing times can be written as

$$J(s)_{LB_2} = \sum_{w(s)=1}^{w_{\max}(s)} \sum_{i=cl_{w(s)-1}+1}^{cl_{w(s)}} w(s)\, p_{[n_{US(s)}-i+1]} \tag{4.18}$$

where $p_{[n_{US(s)}-i+1]}$ denotes the processing time of i^{th} largest unscheduled job.

If we let $w(s)_{[i]}$ be the i^{th} element in the sorted set of weights at the current node, then $J(s)_{LB_2}$ can simply be expressed as

$$J(s)_{LB_2} = \sum_{i=1}^{n_{US(s)}} w(s)_{[i]} p_{[n_{US(s)}-i+1]} \tag{4.19}$$

Combining $M(s)_{LB_2}$ and $J(s)_{LB_2}$, we obtain the following expression for $LB_2(s)$:

$$\begin{aligned} LB_2(s) &= F(s) + nub_{k(s)} C_{j(s)} + \sum_{k=k(s)+1}^{h(s)} \sum_{r=1}^{k-1} nub_k (T_r + t_r) + \sum_{w(s)=1}^{w_{\max}(s)} \sum_{i=cl_{w(s)-1}+1}^{cl_{w(s)}} w(s) p_{[n_{US(s)}-i+1]} \\ &= F(s) + nub_{k(s)} C_{j(s)} + \sum_{k=k(s)+1}^{h(s)} \sum_{r=1}^{k-1} nub_k (T_r + t_r) + \sum_{i=1}^{n_{US(s)}} w_{[i]}(s) p_{[n_{US(s)}-i+1]} \end{aligned} \tag{4.20}$$

where $F(s)$ is the total flow time of the scheduled jobs of s.

We fathom s if $LB_2(s)$ is no less than the best known upper bound.

We let UB_1 and UB_2 denote the total flow time values of heuristics ISPT and MSPT respectively. We use $\min\{UB_1, UB_2\}$ as an initial feasible solution.

Whenever a batch has to be closed, we compute an upper bound assuming that the remaining jobs are scheduled by the SPT rule. We update the initial upper bound whenever the upper bounds at the nodes are smaller or whenever a complete solution with smaller total flow time value is found.

If a partial schedule cannot be fathomed by the optimality properties or the lower bound mechanisms, then we add it to the sorted set of child nodes of the corresponding level. As discussed, we use depth first strategy to explore the search tree. At any level $l \le n$, we choose the child node having the lowest lower bound, i.e., the most promising node, to branch. Other child nodes are added to the set of active nodes. After a complete schedule is obtained at level n, we backtrack to level $n-1$ and continue with the most promising child of level $n-1$. To generalize, when all nodes at level l are explored, we backtrack to level $l-1$, and continue branching from this level.

We terminate when there is no unfathomed node. The best known upper bound is the optimal solution.

CHAPTER 5

COMPUTATIONAL RESULTS

In this chapter, we discuss the results of our computational study performed to evaluate the performance of our reduction mechanisms, bounding procedures and branch and bound algorithm. We first describe the generation of our problem data and our performance measures in Sections 5.1 and 5.2, respectively. Then, we present and discuss the results of the experiments in Section 5.3.

5.1 Design of Experiments

We generate several problem combinations using the parameters listed below:

1. Processing Times: The processing times are randomly drawn from a discrete uniform distribution over $[1, A]$. We consider two levels for A, i.e., we use two sets of processing times in our experiments. In the first set, called Set I, $A = 10$, i.e., $p_j \sim \mathrm{U}[1,10]$, and in the second set, called Set II, $A = 100$, i.e., $p_j \sim \mathrm{U}[1,100]$.

2. Problem Size: The problem size is defined by the number of jobs, n. For Set I, we use seven different n values ranging from 20 to 80 in increments of 10. For Set II, we observed that the size of the largest problem instances that can be

solved is smaller, and use seven different n values ranging from 20 to 50 in increments of 5.

3. Maintenance Patterns: There are several maintenance patterns applicable to manufacturing environments and we use four different maintenance patterns in our experiments.

i. Periodic Pattern: One of the commonly used maintenance schemes is the periodic maintenance, i.e., the identical maintenance tasks, i.e., $t_k = t$ for all k, are scheduled periodically, i.e., $T_k = T$ for all k. The maintenance period T and the maintenance duration t are the parameters that define a periodic pattern. We use three levels of T, which are $A, 2A, 3A$, for each processing times set, and we use two levels of t, which are 0 and $0.4T$, for each level of T. The values used for the parameters T and t in our computational study are given in Table 5.1.

Table 5.1 T and t Values of Periodic Patterns

	Set I: p_j~U[1,10]						**Set II: p_j~U[1,100]**					
T	10		20		30		100		200		300	
t	0	4	0	8	0	12	0	40	0	80	0	120

ii. Repeating Pattern I: The maintenance periods are fixed, i.e., $T_k = T$ for all k, and the maintenance durations follow the repeating pattern $0.2T, 0.4T, 0.6T$, i.e., $t_1 = 0.2T, t_2 = 0.4T, t_3 = 0.6T, t_4 = 0.2T, t_5 = 0.4T, t_6 = 0.6T$ and so on. In our experiments on repeating pattern I, the maintenance period lengths are $T = 2A$ for each set of processing times, and the maintenance durations follow the corresponding $0.2T, 0.4T, 0.6T$ pattern. T and t_k values for repeating pattern I in our computational study are given in Table 5.2.

Table 5.2 T and t_k Values of Repeating Pattern I

	Set I: p_j~U[1,10]	**Set II: p_j~U[1,100]**
T	20	200
t_k	4,8,12,4,8,12…	40,80,120,40,80,120…

iii. Repeating Pattern II: The maintenance durations are fixed, i.e., $t_k = t$ for all k and the maintenance period lengths follow the repeating pattern $3A, 2A, A$,

i.e., $T_1 = 3A, T_2 = 2A, T_3 = A, T_4 = 3A, T_5 = 2A, T_6 = A$ and so on. In our experiments on repeating pattern II, the maintenance durations, t, are set to 0. T_k values for repeating pattern II are given in Table 5.3.

Table 5.3 T_k Values of Repeating Pattern II

	Set I: p_j~U[1,10]	**Set II: p_j~U[1,100]**
T_k	30,20,10,30,20,10…	300,200,100,300,200,100…

iv. Random Pattern: Commonly, preventive maintenance activities are scheduled tasks and they usually follow a special pattern such as periodic pattern or one of the repeating patterns discussed above. On the other hand, the random pattern represents an environment where the machine unavailability may be due to the non-maintenance activities, such as pre-planned orders or pre-known material shortages. For the random patterns used in our experiments, the availability period lengths and the unavailability durations are randomly generated from the discrete uniform distributions in the ranges $[A, 3A]$ and $[1, A]$, respectively, for each processing times set.

Table 5.4 T_k and t_k Values of Random Pattern

	Set I, p_j~U[1,10]	**Set II, p_j~U[1,100]**
T_k	T_k ~U[10,30]	T_k ~U[100,300]
t_k	t_k ~U[1,10]	t_k ~U[1,100]

For each processing times set, problem size (n) and maintenance pattern (T_k, t_k) combination, 10 problem instances are considered. For each problem instance, the execution is terminated whenever the optimal solution is not found within the time limit of one hour.

Branch and bound algorithm is coded in Visual C++ 6.0 and all computational experiments are conducted on Intel Pentium IV 2.80 GHz CPU personal computers with 1 GB RAM under the Windows XP operating system.

We also try to solve some small-sized problem instances by the CPLEX optimization software. However, CPLEX software failed to solve the majority of the problem instances with 20 jobs in our termination limit of one hour.

5.2 Performance Measures

In evaluating the performance of our bounding procedures, we use the percentage deviation from the optimal solution as a ratio of the optimal solution as a performance measure:

- To evaluate the performance of a lower bound, LB, we use the average and maximum value of its percentage gap as a ratio of the optimal solution, OPT, for the cases where OPT is known. Hence, our measure is $\frac{OPT - LB}{OPT} * 100$

- To evaluate the performance of an upper bound, UB, we use the average and maximum value of its percentage gap as a ratio of the optimal solution, OPT, for the cases where OPT is known. Hence, our measure is $\frac{UB - OPT}{OPT} * 100$

In evaluating the performance of our branch and bound algorithm, we use the following performance measures:

- *Computation Time:* Average and maximum values of computation times in Central Processing Unit (CPU) seconds are reported for each problem setting. Average computation times are calculated with unsolved instances contributing one hour to the total computation times.

- *Number of Nodes:* Average and maximum number of nodes explored in branch and bound tree are reported for each problem setting. Unsolved instances are also considered in computing the average number of nodes and determining the maximum number of nodes.

- *Optimality Node (Best Node Number):* Average and maximum number of nodes generated till the optimal solution is reached are reported for each problem setting. Unsolved instances are also considered and number of nodes till the best solution is reached are used for those instances.

- *Number of Unsolved Problem Instances:* For each problem setting, we report the number of problem instances, out of 10, that cannot be solved in one hour.

5.3 Discussion of the Results

5.3.1 Effect of the Reduction and Bounding Mechanisms

To investigate the performance of our reduction and bounding mechanisms, we consider 10 instances for each of the following 4 problem settings:

- Periodic pattern, $T = 2A,\ t = 0.4T$
  - Set I, $n = 50$
  - Set II, $n = 35$
- Random pattern, $T_k \sim \mathrm{U}[A, 3A],\ t_k \sim \mathrm{U}[1, A]$
  - Set I, $n = 50$
  - Set II, $n = 35$

Hence, we consider 40 problem instances. For these problem settings we compare the performance of the branch and bound algorithm utilizing all the mechanisms discussed and with the performance of branch and bound algorithm excluding only mechanism whose effect is of concern.

Note that, the random pattern has an expected T value of $2A$ which is the T value of the periodic pattern considered. Our preliminary experiments have revealed that the effect of t on the performance of the algorithm is negligible compared to that of T. So, one can expect similar performances for the selected settings.

a) Effect of the Reduction Mechanisms

We now discuss the effect of using the reduction mechanisms in our branch and bound algorithm. The first reduction mechanism corresponds to condition (b) and the second reduction mechanism corresponds to conditions (c) and (d) (see Section 4.5). To observe their effect, we perform experiments on the branch and bound algorithm

- incorporating all mechanisms,
- excluding only the reduction mechanism 1,
- excluding only the reduction mechanism 2,
- using no reduction mechanisms.

As can be seen from Table 5.5, both mechanisms have a significant effect on the performances of our branch and bound algorithm. The smallest effect on

CPU times is seen on the random pattern for Set I problems. The average CPU time of these problems is nearly doubled when both reduction mechanisms are ignored. The problem instances that are affected by the reduction mechanisms at the greatest extent are from the periodic pattern and Set II processing times. Their average CPU time is almost increased by fifteen times and one of the problem instances remain unsolved solved within our time limit when no reduction mechanism is used. When we exclude the worst performing instance, we observe that the increase is not as significant, and the average CPU time of the remaining instances are almost doubled for this problem combination as well.

It should be noted that, for some combinations incorporating the reduction mechanisms increases the CPU times. For these combinations, the effort used to check the mechanisms is more than the savings brought.

Although both reduction mechanisms are notably effective, when we compare the results of excluding only reduction mechanism 1 and only reduction mechanism 2 only, we observe that the first mechanism is more effective. We observe that the enormous difference between the effects of the reduction mechanisms is mainly caused by a single instance and excluding the worst performing instance of the periodic pattern problems with Set II processing times reduces the difference between the performances of the reduction mechanisms.

The results given on Table 5.5 also reveal that reduction mechanism 2 is not as effective in random pattern problems as in periodic pattern problems. This is because the feasible exchange of two batches occurs rarely in random pattern. Thereof, excluding reduction mechanism 2 can decrease CPU time as it does not eliminate the nodes in some instances.

Excluding reduction mechanisms increases the average and maximum number of nodes and optimality node distinguishably. Reduction mechanism 1 is more effective than reduction mechanism 2 on the number of nodes and optimality node. We also observe that excluding mechanism 2 increases the number of nodes and optimality node only in few instances. This can be attributed to the elimination of nodes which lead to alternative optimal solutions. Eliminating such nodes causes updating the upper bound later, and hence, using reduction mechanism 2 increases the number of nodes.

Table 5.5 The Computational Results of the Branch and Bound Algorithm with and without Reduction Mechanisms

	Patterns	Set Number	n	CPU Time (Sec.)		Number of Nodes		Optimality Node	
				Avg	Max	Avg	Max	Avg	Max
With Reduction Mechanisms 1 and 2	Periodic T=2A t=0.4T	I	50	22.01	142.13	2,242,607	14,496,812	109,207	740,226
		II	35	26.47	193.41	4,992,198	37,392,046	398,472	3,531,865
	Random T_k~U[A,3A] t_k~U[1,A]	I	50	29.11	257.48	1,997,253	17,723,944	14,948	93,170
		II	35	5.42	34.92	697,772	4,650,661	76,676	397,805
Without Reduction Mechanism 1	Periodic T=2A t=0.4T	I	50	50.35	297.14	5,060,772	29,856,227	321,097	2,171,572
		II	35	395.03*	3600.00	77,174,285	708,520,179	15,579,553	151,448,522
	Random T_k~U[A,3A] t_k~U[1,A]	I	50	45.35	329.50	3,745,040	28,501,582	47,985	404,254
		II	35	44.05	322.60	5,264,134	40,204,183	510,331	3,477,898
Without Reduction Mechanism 2	Periodic T=2A t=0.4T	I	50	36.96	191.44	3,724,736	19,417,395	243,291	1,718,930
		II	35	55.36	463.31	10,153,004	85,760,462	1,108,439	10,328,447
	Random T_k~U[A,3A] t_k~U[1,A]	I	50	30.53	271.34	2,127,901	18,943,561	16,723	111,561
		II	35	5.30	34.09	697,990	4,652,728	76,697	397,805
Without Reduction Mechanisms 1 and 2	Periodic T=2A t=0.4T	I	50	72.43	385.21	6,906,188	36,975,375	574,385	3,476,153
		II	35	409.68*	3600.00	70,991,907	627,377,970	43,630,781	426,512,880
	Random T_k~U[A,3A] t_k~U[1,A]	I	50	49.39	359.74	4,147,294	32,501,582	51,567	438,025
		II	35	45.09	319.63	5,418,333	40,225,872	662,366	4,997,992

(*) 1 out of 10 instances of the corresponding problem combination could not be solved within one hour time limit.

b) Effect of the Upper Bounds

As can be seen from Table 5.6, maximum and average number of total nodes explored and number of nodes explored till the optimal solution is found, optimality node, decrease when the upper bounding mechanisms are used. This result is due to the fact that, upper bounding mechanisms facilitate finding a better feasible solution and updating the best known solution more frequently. So, many more nodes are eliminated as we have no worse upper bound at any partial schedule when the upper bounding mechanisms are incorporated.

The greatest increase in the number of nodes and optimality node when the upper bounds are excluded is observed in problems with random pattern where $n = 35$ and $p_j \sim \text{U}[1,100]$. Both performance measures are doubled in those problems when upper bounding mechanisms are excluded. It should be pointed out that this increase is due to the worst performing instance among the 10 problem instances, i.e., the instance having maximum number of nodes and optimality node. When we exclude this instance, we observe that the increases in performance measures are not as significant. For the remaining nine instances, the average number of nodes and optimality node are 711,461 and 53,940, respectively when all mechanisms are included, and these two performance measures raise to 731,840 and 73,829, respectively, when the upper bounds are excluded.

We use two simple and fast upper bounding procedures to obtain an initial feasible solution at root node with negligible computation effort. As discussed, we also update the upper bound at nodes corresponding to closing batches by sequencing the unscheduled jobs in SPT order. SPT order is determined while computing the lower bound, and the additional computational effort spent is not too significant. But as there can be too many nodes where batches are closed, the total additional time cannot be ignored. So, we should compare the computation time of the algorithm with and without upper bounding procedures to justify the use of upper bounding mechanisms.

Table 5.6 The Computational Results of the Branch and Bound Algorithm with and without Upper Bounds

	Patterns	Set Number	n	CPU Time (Sec.)		Number of Nodes		Optimality Node	
				Avg	Max	Avg	Max	Avg	Max
With Upper Bounds	Periodic T=2A t=0.4T	I	50	22.01	142.13	2,242,607	14,496,812	109,207	740,226
		II	35	26.47	193.41	4,992,198	37,392,046	398,472	3,531,865
	Random T_k~U[A,3A] t_k~U[1,A]	I	50	29.11	257.48	1,997,253	17,723,944	14,948	93,170
		II	35	5.42	34.92	697,772	4,650,661	76,676	397,805
Without Upper Bounds	Periodic T=2A t=0.4T	I	50	88.96	610.02	9,903,691	71,434,973	162,834	787,812
		II	35	25.45	184.10	5,035,433	37,485,631	422,723	3,625,456
	Random T_k~U[A,3A] t_k~U[1,A]	I	50	115.61	1122.66	7,934,638	77,094,250	22,296	93,926
		II	35	5.75	34.61	760,567	4,650,819	140,089	736,429

As can be observed from Table 5.6, there is a significant increase in the average computation time when the patterns are periodic and random, and $n = 50$ and $p_j \sim U[1,10]$. For these combinations, excluding upper bounds increases the average computation time. Note that the average CPU times increases from 22.01 to 88.96 seconds and from 29.11 to 115.61 seconds, for periodic and random pattern instances respectively. When we investigate these two problem settings, we observe that this huge increase is due to the worst performing instance for one problem setting and due to the two worst performing instances in the other setting. When we exclude these instances, we see that the upper bounds are not that much significant. For the problems with periodic and random patterns where $n = 35$ and $p_j \sim U[1,100]$, there is no significant change in the computation time, when the upper bounds are excluded. There is a slight decrease in the computation time of the periodic pattern problems where $n = 35$ and $p_j \sim U[1,100]$, which can be attributed to the similar reasons we just discussed. On the other hand, there is a slight increase in the computation time of the random pattern problems where $n = 35$ and $p_j \sim U[1,100]$, so it is worthy to update the upper bounds as they eliminate a notable amount of nodes which in turn reduces the computation time.

Note that, the effect of the upper bounds becomes more significant when n increases. When n is small, without upper bounding mechanisms, the upper bound are update more frequently as it is quicker to reach the highest level of the tree, level n.

Another point that should be highlighted is the relative performance of the two upper bounding procedures, ISPT and MSPT, used at the root node. For all 40 problems investigated, the upper bound found by MSPT, i.e., UB_2, is better than the upper bound found by ISPT, i.e., UB_1. As can be observed from Table 5.7, average percentage gap of UB_1 from the optimal solution is about 9%-10%, while the percentage gap of UB_2 from the optimal solution is about 1%-2% for all problem settings. However, in our initial experiments, we observe some instances where ISPT performs better. These occur when the number of batches used is very small and both upper bounding procedures return the same number of batches.

Although, the instances where $UB_1 < UB_2$ is very rare, we prepare to include UB_1 due to its negligible and one-time computational effort.

Table 5.7 Root Node Performances of Upper Bounds

Patterns	Set Number	n	Upper Bound 1 (ISPT)		Upper Bound 2 (MSPT)	
			(UB-OPT)/OPT*100		(UB-OPT)/OPT*100	
			Avg	Max	Avg	Max
Periodic T=2A t=0.4T	I	50	9.28	11.13	2.03	5.33
	II	35	9.56	13.98	1.16	4.57
Random T_k~U[A,3A] t_k~U[1,A]	I	50	9.55	13.18	0.85	2.19
	II	35	10.24	16.48	1.72	4.22

As, additional effort spent to find the upper bounds is less than the savings observed when they are used, we include UB_1, UB_2 and upper bound update at nodes corresponding to closing a batch, in our main experiment.

c) Effect of the Lower Bounds

To investigate the effect of lower bounds on the performance of branch and bound algorithm, we perform experiments by

- incorporating the proposed lower bounds
- excluding only the lower bound 1
- excluding only the lower bound 2
- using no lower bound, i.e., only considering the total flow time of the partial schedule as the lower bound

As can be seen from Table 5.8, maximum and average computation time, number of total nodes explored and optimality node increase significantly when any of the lower bounding mechanisms is ignored. Since no problem instances can be solved within the one hour time limit when both lower bounds are not used, no results are reported for the algorithm using no lower bound.

Table 5.8 The Computational Results of the Branch and Bound Algorithm with and without Lower Bounds

	Patterns	Set Number	n	CPU Time (Sec.)		Number of Nodes		Optimality Node	
				Avg	Max	Avg	Max	Avg	Max
With Lower Bounds 1 and 2	Periodic T=2A t=0.4T	I	50	22.01	142.13	2,242,607	14,496,812	109,207	740,226
		II	35	26.47	193.41	4,992,198	37,392,046	398,472	3,531,865
	Random T_k~U[A,3A] t_k~U[1,A]	I	50	29.11	257.48	1,997,253	17,723,944	14,948	93,170
		II	35	5.42	34.92	697,772	4,650,661	76,676	397,805
Without Lower Bound 1	Periodic T=2A t=0.4T	I	50	25.86	165.97	2,735,499	17,425,789	136,907	951,531
		II	35	44.35	317.79	8,516,509	60,711,284	616,719	5,444,081
	Random T_k~U[A,3A] t_k~U[1,A]	I	50	32.57	288.69	2,246,128	19,954,460	16,569	106,016
		II	35	7.50	47.08	983,760	6,332,906	96,674	493,665
Without Lower Bound 2	Periodic T=2A t=0.4T	I	50	96.63	565.61	11,230,855	66,493,140	165,268	807,966
		II	35	257.45	1399.76	51,471,899	291,061,515	1,757,005	7,231,706
	Random T_k~U[A,3A] t_k~U[1,A]	I	50	282.72	2262.82	20,526,510	165,203,317	254,279	1,225,311
		II	35	189.68	979.39	25,532,146	137,109,800	1,684,095	12,774,747

We observe that either of the lower bounds has a dominant effect on the performance measures in all problems settings, and significant effect is when $p_j \sim U[1,100]$. For example, ignoring lower bound 1, increases the average CPU time of the Set II problems with processing times and periodic pattern from 26.47 seconds to 44.35 seconds. The greatest percentage increase observed in CPU time without lower bound 2 corresponds to the Set II problems with random pattern. The average CPU time of the associated instances increases from 5.42 seconds to 189.68 seconds, i.e., approximately 35 times. Similar effects are seen on the other performance measures, i.e., excluding any one of the lower bounds increases the number of nodes and optimality node, and the greatest increase is associated to the Set II problems.

The computational results given in Table 5.8 reveal that lower bound 2 is more effective than lower bound 1 as excluding it deteriorates the performance more significantly. Although, no problem instances are solved with any lower bounds, excluding only one of them does not leave any problem instance unsolved within our time limit of one hour. This can be attributed to the nodes which can be eliminated by both either of the lower bounds, i.e., $LB_1(s) \geq UB$ and $LB_2(s) \geq UB$.

We also observe that lower bound 2 performs better than lower bound 1 at the root node in all 40 problem instances. As can be seen from Table 5.9, average percentage gap of lower bound 1 at the root node, LB_1, from the optimal solution is about 2%-4%, while the average percentage gap of the lower bound 2 at the root node, LB_2, from the optimal solution is about 1%-2% for the 4 problem settings. Although $LB_1 < LB_2$ in vast majority of the instances, it is not reasonable to exclude lower bound 1 from the branch and bound algorithm since LB_1 is anyway necessary for computing LB_2.

Table 5.9 Root Node Performances of Lower Bounds

Patterns	Set Number	n	Lower Bound 1 (PSPT)		Lower Bound 2 (Matching)	
			(OPT-LB)/OPT*100		(OPT-LB)/OPT*100	
			Avg	Max	Avg	Max
Periodic T=2A t=0.4T	I	50	2.58	3.74	1.58	3.29
	II	35	3.50	4.88	1.76	3.65
Random T_k~U[A,3A] t_k~U[1,A]	I	50	2.08	3.36	1.35	2.75
	II	35	3.07	4.21	1.34	2.80

In order to observe the effect of excluding both lower bounds on the performance measures, we use small size problem instances with 20 jobs and report the computational results in Table 5.10. Although, the average computation time for all problem settings are no greater than 0.02 seconds when at least one of the lower bounds are used, excluding both of them increases the computation time drastically. As discussed, lower bounds are more effective for the problems of Set II. Note that, ignoring lower bounds 1 and 2, increases the average CPU time of the periodic and random pattern problem instances of Set II from a negligible amount of time, i.e., 0.00 seconds, to 239.54 and 1472.50 seconds, respectively. Two instances of the random pattern problems cannot be solved within the time limit.

To summarize, lower bounding is the most effective mechanism of the branch and bound algorithm. Optimal solution for any of the large-sized problem instances, i.e., not less than 50 and 35 jobs for Set I and Set II problems, respectively, and for some small-sized problem instances cannot be obtained within the time limit of one hour when both lower bounds are excluded.

Table 5.10 The Computational Results of the Branch and Bound Algorithm with and without Lower Bounds for $n = 20$

	Patterns	Set Number	CPU Time (Sec.)		Number of Nodes		Optimality Node	
			Avg	Max	Avg	Max	Avg	Max
With Lower Bounds 1 and 2	Periodic T=2A t=0.4T	I	0.00	0.00	133	458	100	371
		II	0.00	0.02	1,087	3,100	439	2,620
	Random T_k~U[A,3A] t_k~U[1,A]	I	0.00	0.02	180	575	60	94
		II	0.00	0.02	509	1,072	289	888
Without Lower Bound 1	Periodic T=2A t=0.4T	I	0.00	0.00	193	561	140	434
		II	0.01	0.02	1,816	4,955	699	4,132
	Random T_k~U[A,3A] t_k~U[1,A]	I	0.00	0.02	256	833	94	161
		II	0.00	0.02	899	1,599	510	1,468
Without Lower Bound 2	Periodic T=2A t=0.4T	I	0.01	0.03	663	1,623	244	955
		II	0.02	0.08	6,862	20,411	1,403	7,167
	Random T_k~U[A,3A] t_k~U[1,A]	I	0.00	0.02	615	1,328	148	683
		II	0.02	0.06	3,264	10,908	1,391	4,103
Without Lower Bounds 1 and 2	Periodic T=2A t=0.4T	I	2.32	8.44	791,450	2,926,637	4,337	22,718
		II	239.54	649.39	82,655,235	219,391,760	138,110	1,216,751
	Random T_k~U[A,3A] t_k~U[1,A]	I	16.14	61.13	5,132,568	19,036,148	166,954	1,642,142
		II	1472.50*	3600.00	489,585,529	1,234,960,486	1,479,953	10,362,369

(*) 2 out of 10 instances of the corresponding problem combination could not be solved within one hour time limit.

5.3.2 Lower Bound Performances

The lower bound ($\max\{LB_1, LB_2\}$) performances, i.e., the average and maximum percent deviations from the optimal solution, for the periodic pattern and other patterns are reported on Tables 5.11 and 5.12, respectively.

Table 5.11 Lower Bound Performances – Periodic Pattern

n	T	t	Set I (OPT-LB)/OPT*100		n	T	t	Set II (OPT-LB)/OPT*100	
			Avg	Max				Avg	Max
20	10	0	7.15	12.32	20	100	0	6.72	11.39
		4	8.35	14.85			40	7.94	14.24
	20	0	0.81	3.45		200	0	2.12	7.21
		8	0.77	4.07			80	2.33	8.94
	30	0	0.85	1.19		300	0	0.85	2.88
		12	0.67	0.96			120	0.81	3.58
30	10	0	4.78	9.10	25	100	0	6.32	10.64
		4	5.37	10.60			40	7.27	12.61
	20	0	1.19	2.76		200	0	1.68	3.70
		8	1.12	2.91			80	1.72	3.98
	30	0	1.13	2.06		300	0	0.87	2.48
		12	0.99	2.09			120	0.82	2.72
40	10	0	6.90	11.99	30	100	0	6.36	8.31
		4	7.77	14.09			40	7.24	9.58
	20	0	1.98	3.47		200	0	2.01	2.97
		8	1.97	3.72			80	2.12	3.08
	30	0	0.90	1.32		300	0	1.14	1.90
		12	0.71	1.25			120	1.00	2.03
50	10	0	7.64	13.31	35	100	0	6.58(3)	9.03
		4	8.59	15.78			40	7.47(3)	10.52
	20	0	1.63	3.22		200	0	1.78	3.50
		8	1.62	3.41			80	1.81	3.79
	30	0	1.24	2.33		300	0	1.07	2.18
		12	1.13	2.42			120	0.87	2.18
60	20	0	1.50(3)*	2.24	40	200	0(2)	1.63(2)	2.77
		8	1.61(2)	2.96			80(2)	1.60(2)	2.86
	30	0	1.17	1.89		300	0	1.07	2.11
		12	1.01	1.85			120	0.98	2.24
70	30	0	1.28	1.71	45	300	0	1.05	1.59
		12	1.16	1.63			120	0.87	1.44
80	30	0	1.35(3)	1.67	50	300	0	1.33(2)	1.62
		12	1.24(3)	1.53			120	1.22(2)	1.65

(*)The numbers in parentheses denote the number of unsolved instances, out of 10, within one hour time limit.

Table 5.12 Lower Bound Performances – Repeating and Random Patterns

	Repeating Patterns				**Random Patterns**	
	Repeating Pattern 1		**Repeating Pattern 2**			
	T=2A t_k**=0.2A,0.4A,0.6A**		T_k**=3A,2A,A** **t=0**		T_k **~U[A, 3A]** t_k **~U[1,A]**	
n	**Set I**					
	(OPT-LB)/OPT *100		**(OPT-LB)/OPT *100**		**(OPT-LB)/OPT *100**	
	Avg	**Max**	**Avg**	**Max**	**Avg**	**Max**
20	0.75	3.72	1.10	3.06	1.41	4.61
30	1.11	2.91	1.41	2.19	1.43	3.17
40	1.99	3.72	1.65	3.23	1.62	2.47
50	1.62	3.45	1.51	2.76	1.35	2.75
60	1.60(2)*	2.83	1.49	2.32	1.38	2.15
n	**Set II**					
	(OPT-LB)/OPT*100		**(OPT-LB)/OPT*100**		**(OPT-LB)/OPT*100**	
	Avg	**Max**	**Avg**	**Max**	**Avg**	**Max**
20	2.17	8.08	1.51	3.43	1.20	3.32
25	1.71	4.21	1.16	2.41	1.32	5.25
30	2.06	3.07	1.94	3.12	1.65	2.89
35	1.79	3.72	1.22	2.18	1.34	2.80
40	1.61(2)	2.96	1.59	3.01	1.15	2.74
45	2.04(5)	2.50	1.47	1.98	1.37(1)	2.28
50	-	-	1.54(2)	1.93	1.29(4)	2.14

(*)The numbers in parentheses denote the number of unsolved instances, out of 10, within one hour time limit.

As can be seen from Table 5.11, in general the lower bounds perform better for small n. For example, when $T = 10$, $t = 4$ and $p_j \sim$ U[1,10], average deviations from the optimal solution are 5.37% and 7.77% for $n = 30$ and $n = 40$, respectively. However, there are some combinations where lower bound performs better when n is smaller. For example, when $T = 100$, $t = 0$ and $p_j \sim$ U[1,100], average deviations are 6.72% and 6.32% for $n = 20$ and $n = 25$, respectively. We also analyze the objective function values and observe small magnitudes when n is small. Hence, we can attribute higher deviations at smaller n to those smaller magnitudes. Clearly, when the total flow time is significantly smaller, the deviation as a percentage of the optimal solution is larger although the absolute deviation may not be.

The lower bound performs better when variance of processing times is smaller, when p_j ~ U[1,10]. For example, the average deviations with 30 jobs are between 0.99% and 5.37% when p_j ~ U[1,10] and between 1.00% and 7.24% when p_j ~ U[1,100].

As the results in Table 5.11 reveal, lower bound performances are better when maintenance periods, T, are larger. It can be seen from the table that the average deviation is above 4.5% when $T = A$ and below 1.5% when $T = 3A$.

We also investigate the effect of maintenance duration, t, on the lower bound performances. We observe that increasing maintenance durations deteriorate the performance of the lower bounds when maintenance period lengths are shorter, i.e., when $T = A$. When the maintenance periods are longer, the maintenance duration has no significant effect on the lower bound performances. This is due to the fact that, there are more batches when T is smaller, hence the contribution of the maintenance durations to the total flow time is higher.

As can be seen from Table 5.12, the lower bound performances are quite satisfactory for all problem combinations. For example, when $n = 50$, the overall maximum deviation is below 4% and the overall average deviation is below 2%. The lower bounds perform slightly better for the repeating pattern 2 compared to the repeating pattern 1. The lower bound performances for random pattern are slightly better than repeating patterns for some problem combinations and slightly worse for the other combinations. This can be attributed to the randomness of the maintenance period lengths and maintenance durations.

5.3.3 Upper Bound Performances

The upper bound ($\min\{UB_1, UB_2\}$) performances, i.e., the average and maximum percent deviations from the optimal solution, for the periodic pattern and all other patterns are reported on Tables 5.13 and 5.14, respectively.

Table 5.13 Upper Bound Performances – Periodic Pattern

n	T	t	Set I (UB-OPT/OPT*100		n	T	t	Set II (UB-OPT/OPT*100	
			Avg	Max				Avg	Max
20	10	0	3.02	10.41	20	100	0	7.44	18.45
		4	3.09	10.87			40	7.80	19.45
	20	0	0.62	2.19		200	0	1.04	3.32
		8	0.54	2.31			80	1.06	3.59
	30	0	0.59	2.07		300	0	0.87	2.42
		12	0.46	1.66			120	0.71	1.93
30	10	0	2.72	7.67	25	100	0	5.20	14.12
		4	2.77	8.00			40	5.38	14.69
	20	0	1.17	5.06		200	0	1.23	5.17
		8	1.13	5.31			80	1.24	5.49
	30	0	0.29	0.72		300	0	0.80	2.12
		12	0.23	0.55			120	0.69	2.33
40	10	0	4.50	8.03	30	100	0	5.88	10.76
		4	4.58	8.27			40	6.07	11.08
	20	0	1.91	6.30		200	0	0.95	2.88
		8	1.96	6.55			80	0.93	3.07
	30	0	1.30	3.07		300	0	0.95	3.12
		12	1.32	3.33			120	0.95	3.43
50	10	0	4.57	7.17	35	100	0	6.59(3)	9.13
		4	4.64	7.27			40	6.78(3)	9.35
	20	0	1.97	5.16		200	0	1.16	4.18
		8	2.03	5.33			80	1.16	4.57
	30	0	0.95	2.75		300	0	0.77	2.22
		12	0.93	2.83			120	0.71	2.39
60	20	0	1.64(3)*	4.58	40	200	0	0.88(2)	2.59
		8	1.96(2)	4.80			80	0.87(2)	2.82
	30	0	0.56	1.80		300	0	0.93	4.46
		12	0.53	1.86			120	0.88	4.83
70	30	0	1.32	3.07	45	300	0	0.56	2.35
		12	1.35	3.23			120	0.50	2.51
80	30	0	1.29(3)	2.48	50	300	0	1.32(3)	2.29
		12	1.33(3)	2.58			120	1.31(3)	2.44

(*)The numbers in parentheses denote the number of unsolved instances, out of 10, within one hour time limit.

Table 5.14 Upper Bound Performances – Repeating and Random Patterns

	Repeating Patterns				**Random Patterns**	
	Repeating Pattern 1		**Repeating Pattern 2**			
	T=2A t_k=0.2A,0.4A,0.6A		**T_k=3A,2A,A t=0**		**T_k ~U[A, 3A] t_k ~U[1,A]**	
n	**Set I**					
	(UB-OPT)/OPT *100		**(UB-OPT)/OPT *100**		**(UB-OPT)/OPT *100**	
	Avg	**Max**	**Avg**	**Max**	**Avg**	**Max**
20	0.59	2.73	1.10	4.51	0.27	0.64
30	1.16	5.59	1.20	2.65	1.41	3.73
40	1.96	6.59	1.10	3.05	2.09	3.35
50	2.06	5.41	1.30	2.84	0.85	2.19
60	1.98(2)*	4.86	1.31	2.13	1.44	2.73
n	**Set II**					
	(UB-OPT)/OPT *100		**(UB-OPT)/OPT *100**		**(UB-OPT)/OPT *100**	
	Avg	**Max**	**Avg**	**Max**	**Avg**	**Max**
20	1.16	4.27	1.11	2.56	1.61	4.18
25	1.24	5.42	1.78	5.39	1.24	4.74
30	0.94	3.16	2.12	4.32	1.12	3.36
35	1.20	4.69	1.85	2.94	1.72	4.22
40	0.86(2)	2.77	0.73	1.42	1.21	2.17
45	0.80(5)	1.67	0.74	1.52	1.44(1)	2.73
50	-	-	1.37(2)	2.37	0.81(4)	1.75

(*)The numbers in parentheses denote the number of unsolved instances, out of 10, within one hour time limit.

When maintenance periods are small, i.e., $T = A$, in general, the upper bounds perform better for small n in general although there are some exceptions due to small total flow time values for small n. We observe that n has no significant effect on the upper bound performances when T is larger, and the upper bounds perform quite well for all values of n for those problems. For example, the average deviation is below 1.5% for all n values when $T = 3A$.

When the variance of the processing times is smaller, the upper bounds perform better, i.e., when $p_j \sim \mathrm{U}[1,10]$. For example, when $T = A$ and $t = 0$, the average deviations with 10 jobs are 3.02% and 7.44% for $p_j \sim \mathrm{U}[1,10]$ and $p_j \sim \mathrm{U}[1,100]$, respectively. For the same problem combinations, the maximum deviations are 10.41% and 18.45% for $p_j \sim \mathrm{U}[1,10]$ and $p_j \sim \mathrm{U}[1,100]$, respectively.

As can be observed from Table 5.13, the upper bound performances are better when the maintenance periods, T, are larger. For example, the average deviation is above 2.5% when $T = A$ and below 1.5% when $T = 3A$.

The effect of the maintenance durations, t, on the upper bound performances is similar to their effect on the lower bound performances. Increasing the durations of maintenance tasks worsens the performance of upper bounds when maintenance periods are smaller, i.e., when $T = A$. The effect of maintenance durations becomes insignificant when the length of the maintenance periods increases. This effect is due to the greater contribution of the maintenance durations to the total flow time when the maintenance period is smaller as there are more batches in such patterns.

It is observed from Table 5.14 that the upper bounds perform slightly better for the repeating pattern 2 compared to repeating pattern 1 for $p_j \sim \mathrm{U}[1,10]$, and their performance for the repeating pattern 1 becomes better than for the repeating pattern 2 when the variance of the processing times increase, i.e., when $p_j \sim \mathrm{U}[1,100]$. The upper bound performances for the random pattern are better than those of the repeating patterns for some problem combinations, and worse for the other problem combinations, i.e., there is no consistent behavior of the random pattern over the repeating patterns.

5.3.4 Branch and Bound Performances

Finally, we discuss the performances of our branch and bound algorithm. Computational results for Set I problems with the periodic pattern and the other patterns are reported on Tables 5.15 and 5.16, respectively. The respective results for Set II problems with these patterns are given in Tables 5.17 and 5.18.

Table 5.15 Branch and Bound Performances – Periodic Pattern, Set I

n	T	t	CPU Time (Sec.)		Number of Nodes		Optimality Node	
			Avg	Max	Avg	Max	Avg	Max
20	10	0	0.01	0.02	2,725	5,528	249	814
		4	0.01	0.02	2,715	5,406	249	814
	20	0	0.00	0.00	141	465	105	374
		8	0.00	0.00	133	458	100	371
	30	0	0.00	0.02	77	150	52	85
		12	0.00	0.02	71	140	48	77
30	10	0	0.47	1.59	106,276	339,124	4,215	20,060
		4	0.48	1.61	106,305	341,120	4,230	20,167
	20	0	0.02	0.11	3,410	20,354	259	817
		8	0.02	0.11	3,244	19,983	247	803
	30	0	0.00	0.02	349	1,320	107	169
		12	0.00	0.02	315	1,195	99	156
40	10	0	14.19	38.41	2,483,991	6,688,762	141,851	717,610
		4	14.25	39.42	2,489,912	6,865,530	142,451	719,496
	20	0	1.01	3.87	133,245	509,819	7,676	29,519
		8	0.98	3.80	127,621	497,442	7,447	28,182
	30	0	0.01	0.06	1,799	7,536	1,093	5,164
		12	0.01	0.05	1,573	6,751	1,015	4,757
50	10	0	941.83	1671.73	132,902,328	236,607,499	2,929,077	18,470,119
		4	946.31	1673.41	133,510,871	240,861,589	3,021,860	18,956,105
	20	0	23.06	146.20	2,364,195	15,011,453	112,095	761,842
		8	22.01	142.13	2,242,607	14,496,812	109,207	740,226
	30	0	4.60	36.33	436,671	3,438,266	80,632	786,573
		12	4.48	35.58	423,070	3,348,013	78,257	763,734
60	20	0	1245.74(3)*	3600.00	104,611,465	303,312,877	1,470,687	14,228,195
		8	1203.19(2)	3600.00	100,546,944	299,643,969	1,449,806	14,019,441
	30	0	30.33	164.36	2,307,335	12,293,934	13,661	41,911
		12	27.69	144.61	2,090,055	10,747,040	12,831	41,470
70	30	0	170.37	980.45	10,610,804	61,043,504	161,515	1,298,225
		12	131.59	711.22	8,097,142	43,697,635	123,621	942,334
80	30	0	1610.30(3)	3600.00	86,007,899	192,493,591	1,096,632	9,041,307
		12	1501.11(3)	3600.00	75,216,952	187,253,284	1,067,768	8,789,038

(*)The numbers in parentheses denote the number of unsolved instances, out of 10, within one hour time limit.

Table 5.16 Branch and Bound Performances–Repeating and Random Patterns, Set I

n	Repeating Pattern 1 $T=20$ $t_k=2,4,6$					
	CPU Time (Sec.)		Number of Nodes		Optimality Node	
	Avg	Max	Avg	Max	Avg	Max
20	0.00	0.02	150	622	112	492
30	0.02	0.14	3,981	25,651	247	798
40	1.10	4.88	157,652	719,534	7,547	29,725
50	22.99	149.69	2,740,825	17,819,752	113,310	771,918
60	1196.17(2)*	3600.00	119,579,446	361,263,699	1,518,823	14,703,976

n	Repeating Pattern 2 $T_k=30,20,10$ $t=0$					
	CPU Time (Sec.)		Number of Nodes		Optimality Node	
	Avg	Max	Avg	Max	Avg	Max
20	0.00	0.00	125	287	65	109
30	0.01	0.03	809	3,721	170	425
40	0.54	2.92	50,002	276,714	3,419	20,258
50	8.82	52.09	617,364	3,650,556	3,902	18,783
60	98.03	766.34	5,447,096	42,791,541	151,025	883,562

n	Random T_k ~U[10-30] t_k ~U[1-10]					
	CPU Time (Sec.)		Number of Nodes		Optimality Node	
	Avg	Max	Avg	Max	Avg	Max
20	0.00	0.02	180	575	60	94
30	0.01	0.06	1,973	9,126	223	756
40	0.58	3.62	55,502	365,114	14,006	124,921
50	29.11	257.48	1,997,253	17,723,944	14,948	93,170
60	83.54	512.42	4,221,241	25,139,838	185,083	672,846

(*)The numbers in parentheses denote the number of unsolved instances, out of 10, within one hour time limit.

Table 5.17 Branch and Bound Performances – Periodic Pattern, Set II

n	T	t	CPU Time (Sec.)		Number of Nodes		Optimality Node	
			Avg	Max	Avg	Max	Avg	Max
20	100	0	0.11	0.44	36,962	148,932	2,521	10,578
		40	0.11	0.45	36,828	149,422	2,519	10,492
	200	0	0.00	0.02	1,146	3,255	459	2,741
		80	0.00	0.02	1,087	3,100	439	2,620
	300	0	0.00	0.00	166	328	96	185
		120	0.00	0.02	150	287	85	175
25	100	0	1.64	5.27	497,442	1,552,223	8,141	40,438
		40	1.64	5.22	496,620	1,540,317	8,166	40,300
	200	0	0.10	0.78	22,875	181,021	2,147	13,425
		80	0.09	0.72	20,377	160,409	2,053	12,957
	300	0	0.01	0.03	1,579	4,706	273	584
		120	0.01	0.02	1,436	4,345	248	571
30	100	0	72.62	312.29	20,340,845	86,963,843	2,287,773	17,168,711
		40	72.98	313.98	20,395,497	87,149,581	2,312,124	17,453,948
	200	0	1.08	5.25	220,053	1,080,340	40,427	209,235
		80	0.99	4.70	195,507	911,954	39,214	206,569
	300	0	0.04	0.27	8,447	55,803	1,859	11,231
		120	0.04	0.27	7,204	47,422	1,610	9,573
35	100	0	1548.89(3)*	3600.00	402,455,920	937,914,238	11,205,637	70,048,849
		40	1564.25(3)	3600.00	407,517,181	938,292,587	11,779,646	74,817,739
	200	0	28.53	205.98	5,409,847	39,764,965	419,587	3,717,937
		80	26.47	193.41	4,992,198	37,392,046	398,472	3,531,865
	300	0	4.75	47.07	780,143	7,730,294	4,328	32,115
		120	4.01	39.74	657,982	6,520,565	3,923	28,949
40	200	0	972.03(2)	3600.00	159,822,817	611,559,485	4,258,616	38,488,563
		80	954.04(2)	3600.00	155,858,946	609,360,160	4,302,596	39,239,085
	300	0	21.61	72.41	3,477,506	11,653,655	624,401	3,336,153
		120	21.03	75.36	3,346,122	11,994,428	588,100	3,103,365
45	300	0	27.31	136.89	3,903,908	20,457,146	231,305	927,768
		120	23.77	116.69	3,325,789	17,276,260	201,545	790,446
50	300	0	1157.96(3	3600.00	144,537,495	503,211,200	75,543,967	382,642,235
		120	1149.02(3)	3600.00	139,678,966	489,838,629	67,908,087	352,067,070

(*)The numbers in parentheses denote the number of unsolved instances, out of 10, within one hour time limit.

Table 5.18 Branch and Bound Performances–Repeating and Random Patterns, SetII

n	Repeating Pattern 1 $T=200$ $t_k=20,40,60$					
	CPU Time (Sec.)		Number of Nodes		Optimality Node	
	Avg	Max	Avg	Max	Avg	Max
20	0.00	0.02	1,622	5,174	633	2,626
25	0.11	0.80	25,241	184,594	2,369	13,486
30	1.01	4.16	195,060	765,236	37,538	206,949
35	30.36	201.53	6,010,307	40,724,744	398,600	3,532,329
40	971.71(2)*	3600.00	178,592,716	683,195,285	4,387,583	39,889,232
45	2229.99(5)	3600.00	352,359,790	580,231,758	33,471,117	206,076,271

n	Repeating Pattern 2 $T_k=300,200,100$ $t=0$					
	CPU Time (Sec.)		Number of Nodes		Optimality Node	
	Avg	Max	Avg	Max	Avg	Max
20	0.00	0.02	574	1,764	276	966
25	0.04	0.13	8,333	24,931	1,468	7,839
30	0.37	1.19	47,733	166,805	5,666	22,201
35	0.87	6.66	129,264	1,030,527	7,901	32,189
40	288.47	1822.72	31,779,526	198,853,308	2,438,883	13,982,310
45	754.45	3414.04	78,648,410	361,748,713	10,666,996	54,756,674
50	1046.11(2)	3600.00	96,209,549	329,648,777	38,713,021	233,844,467

n	Random T_k ~U[100-300] t_k ~U[1-100]					
	CPU Time (Sec.)		Number of Nodes		Optimality Node	
	Avg	Max	Avg	Max	Avg	Max
20	0.00	0.02	509	1,072	289	888
25	0.10	0.44	18,091	89,446	1,424	5,214
30	1.30	4.05	181,474	541,412	58,636	234,926
35	5.42	34.92	697,772	4,650,661	76,676	397,805
40	283.23	2795.07	33,703,732	332,968,917	86,282	715,133
45	528.57(1)	3600.00	61,451,316	456,020,803	11,054,198	101,116,896
50	1590.75(4)	3600.00	125,211,994	324,107,476	27,789,035	123,263,693

(*)The numbers in parentheses denote the number of unsolved instances, out of 10, within one hour time limit.

The numbers given in parentheses indicate the number of unsolved instances within one hour time limit. Remember that each unsolved instance contributes to 3600 seconds to the total computation time and these instances are considered while calculating the performance measures.

As can be observed from the tables, the branch and bound algorithm's performance deteriorates as the number of jobs, n, increases. For example, the average CPU time of the periodic pattern - Set I problems with $T = 10$, $t = 4$ is 0.48 seconds for $n = 30$ and 14.25 for $n = 40$. This is due to the increased number alternatives at each level for larger values of n. Another reason is that the performances of the bounds deteriorate as n increases, especially for small values of the maintenance period length, T (see Tables 5.11 and 5.13).

The results given in Tables 5.15 and 5.17 reveal that branch and bound algorithm performs better for larger values of T. For example, the average CPU time of the periodic pattern - Set I problems with $n = 50$, $T = 10$, $t = 4$ is 946.31 seconds, and when maintenance period length is doubled, i.e., $T = 20$, while the other parameters are fixed, the average CPU time decreases to 22.01 seconds. One of the reasons of this result is the better performances of the bounds (see Tables 5.11 and 5.13) for larger values of T. Another reason is the decreased number of alternatives. Note that, fewer batches are constructed when the maintenance period is shorter. As we branch to only higher indexed jobs within the same batch, fewer batches implies less number of nodes to be explored which in turn reduces the CPU times.

We observe that maintenance duration, t, has no significant effect on the branch and bound performances. From Tables 5.15 and 5.17, we see that the computation time and number of nodes slightly increase as t when maintenance period is short, i.e., $T = A$. This can be attributed to the deteriorating performances of our bounds as t increases for these problem settings (see Tables 5.11 and 5.13).

We observe that the branch and bound performances of Set II problems, reported on Tables 5.17 and 5.18, are inferior to those of Set I problems, reported on Tables 5.15 and 5.16. For example, the average CPU times of the problems with random pattern 2 and $n = 40$ jobs are 0.54 and 288.47 seconds for Set I and Set II, respectively. The reasons of the deteriorating performances of our branch and bound algorithm with increasing variance of the processing times, are the increased

number of alternatives due to the increased number of distinct jobs at each level and the slightly inferior performances of our bounds.

Combining the effects of the maintenance period length (T) and processing times (p_j), one can conclude that the most difficult problems are in Set II problems and have $T = A$. For this problem combination, our algorithm can solve problem instances with up to 35 jobs. Similarly, the easiest problems are in Set I problems where $T = 3A$ and our algorithm can solve instances with up to 80 jobs of this problem combination.

We observe from Tables 5.16 and 5.18 that the algorithm's performance with random pattern 1 is worse than that of with random pattern 2. For example, when $n = 50$, the average CPU times of Set I problems with random patterns 1 and 2 are 22.99 seconds and 8.82 seconds, respectively. This is because the earlier periods in random pattern 2 are wider and many more jobs with shorter processing times can fit to these periods.

Note that, in random pattern instances, the expected maintenance period length is $2A$ and the expected maintenance duration is $0.5T$. So, one can expect to observe similar results for random pattern and repeating pattern 2. But, when we compare the results of these two patterns given in Tables 5.16 and 5.18, we see that there is not a consistent pattern of the results. For example, as can be observed from Tables 5.18, for Set II problems, the branch and bound performances of random pattern case are better when $n = 45$ and are worse when $n = 50$, compared to the repeating pattern 2.

We now compare the performances of the random pattern and repeating pattern 1. Similar to the previous case, we observe no consistent behavior of the random pattern over repeating pattern 1. For example, from Tables 5.17, for Set I problems we see that the branch and bound performances are better for repeating pattern 1 when $n = 50$ and are worse when $n = 60$, compared to the random pattern.

We observe from more careful investigation of the results that, when the realized maintenance periods are longer, the branch and bound algorithm performs better and when shorter maintenances are realized, the algorithm performs poorer. Hence, realized values of T, highly affects the performance.

When we compare the results given in Tables 5.15 and 5.17 with those of in Tables 5.16 and 5.18, we observe that the problems of repeating patterns are easier to solve than those of the periodic pattern with $T = A$ and harder to solve than those of the periodic pattern with $T = 3A$. For example, as can be seen from Table 5.17, for Set II problems with $n = 30$ jobs the average CPU times of the periodic pattern problem instances for $T = A$ and $T = 3A$ are about 72.5 and 0.04 seconds, respectively. The average CPU times of the repeating patterns 1 and 2 problems with $n = 30$ again, are 1.01 and 0.37, (see Table 5.18). Moreover, we observe that the branch and bound performances for the repeating pattern 1 and periodic pattern with $T = 2A$ are very similar as the effect of t is not very significant. For example, as can be observed from Table 5.15, for Set I problems with $n = 50$ jobs and $T = 2A$, the average CPU times of the periodic pattern are 23.06 and 22.01 for $t = 0$ and $t = 0.4T$, respectively. From Table 5.16, we see that the average CPU time of the Set I problems with repeating pattern 1 and $n = 50$ jobs is 22.99 seconds.

The effect of the parameters on the number of nodes and optimality node is similar to their effect on the computation time.

We observe from the computational results that, the optimality nodes are significantly small compared to the total number of nodes explored. Hence, a solution found before termination limit, i.e., within the one hour time limit, is likely to perform well, if not optimal.

The worst case performances of our branch and bound algorithm are also affected by the problem parameters. Similar to the average performances, the worst case performances deteriorate with increasing number of jobs (n), increasing variance of the job processing times and decreasing maintenance period lengths (T). We see that the maintenance duration (t) has no significant effect on the worst case performances. We also observe that excluding worst case performances significantly improves the average performances of the algorithm, which reveals the high variation in problem difficulty.

CHAPTER 6

A MAINTENANCE PLANNING PROBLEM WITH MINOR AND MAJOR MAINTENANCES

In this chapter, we study a maintenance planning problem with minor and major maintenance tasks. Minor maintenance activities are of shorter durations and involve some routine operations like cleaning, lubrication, oil changes, and small adjustments. On the other hand, major maintenance activities have longer durations and usually include the activities performed to bring the machine to its "as if new" position. These activities may include tool changeovers, replacement of machine parts, major overhauls and inspection. Major maintenance activities may be essential after the machine is operated for a specified time. In many practical applications, there can be a number of minor maintenances between two consecutive maintenances. We assume there is a single minor maintenance between two consecutive major maintenances, and there are two alternatives for implementation. In the first alternative, A1, there are T time units between any two consecutive maintenances. The minor maintenance has a duration of t_m time units and the major maintenance has a duration of t_M time units. The Gantt chart given in Figure 6.1 illustrates the case.

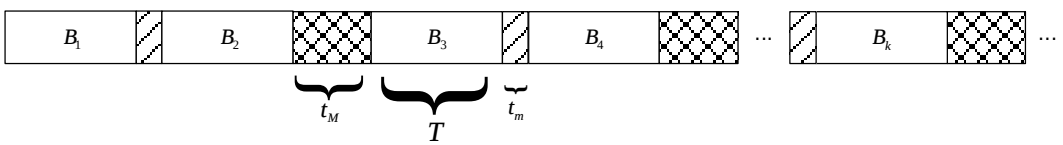


Figure 6.1 Gantt Chart Representation of Minor and Major Maintenances

In real practice, there may be cases where it might be preferred to perform the major maintenance task immediately after the minor one. The main motivation for this preference, is that performing the major maintenance just after the minor one takes shorter time than that of performing it later. This can occur due to the following reasons:

- A portion of the maintenance duration can be attributed to set up type activities such as interrupting the production, preparing the maintenance tools. Hence, combining the major maintenance with the minor one decreases its duration.
- When the major maintenance activity is performed earlier, fewer or less time consuming operations may be required to keep the machine in its operating state. Moreover, minor and major maintenances may have some common operations which will not be duplicated when the major maintenance is performed immediately after the minor one.

Considering the above practical issues, we define our second alternative, A2, where the major maintenance of t'_M is performed just after minor maintenance. We assume $t'_M < t_M$ to justify the earlier performance of a maintenance activity. Figure 6.2 illustrates the schedule where A2 is implemented at all decision points.

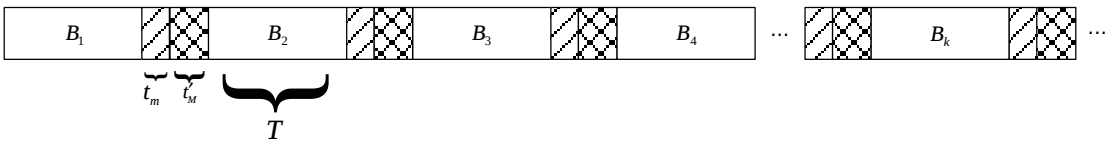


Figure 6.2 The Schedule Having A2 at Each Decision Point

Our maintenance planning and job scheduling problem is to decide on A1 or A2 after each minor maintenance and the jobs in each batch. Figure 6.3 illustrates a feasible solution to the problem where A1 is selected after first and third minor maintenances, and A2 is selected for the rest. In the solution, there are seven batches.

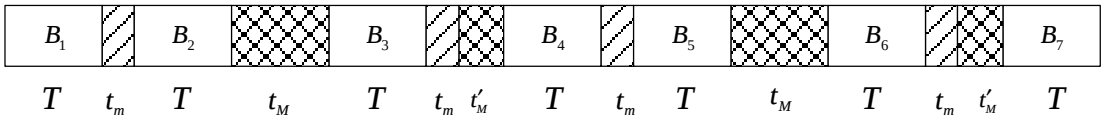


Figure 6.3 A Schedule Having Both A1 and A2 at Different Decision Points

Our aim is to find the job sequence and the maintenance plan with two alternatives A1 and A2, after each minor maintenance. We study the problem for resumable and non-resumable cases, separately.

6.1 Resumable Jobs

For the resumable jobs case, when the minor and major maintenances are pre-planned, the problem reduces to the 1|r-a, multi, fix| $\sum C_j$ problem that is optimally solved by Preemptive Shortest Processing Time (PSPT) rule. (see Section 4.4).

When the maintenance plan is not fixed, after each minor maintenance, there are two choices: immediately perform major maintenance or wait for T time units, i.e., A2 or A1. Hence for a given sequence, there are 2^h decisions if there are h minor maintenances. There are $n!$ choices for sequencing, hence we have a total of $n!.2^h$ choices. However, in this chapter, we show that for a given sequence the optimal maintenance plan can be found in polynomial time without evaluating 2^h choices, and our problem has $n!$ choices. Below is the detailed description of our optimal maintenance plan for a given sequence.

We let RA_k denote the number of jobs that are completed in batch k in the PSPT schedule. Thus, RA_k satisfies $\sum_{j=1}^{RA_k} p_j \le kT$ and $\sum_{j=1}^{RA_k+1} p_j > kT$ given that the jobs are indexed in the SPT order.

Assume that there is a minor maintenance performed just after batch k. If A2 is chosen, then each job in batch $k+1$ will start t'_M time units later, compared to the start times when A1 is chosen. This increases the total flow time by $RA_{k+1} * t'_M$ units. On the other hand, each job to be processed in batches $k+2, k+3, ..., q$ will start $(t_M - t'_M - t_m)$ time units earlier, thereby reducing the total flow time by $(n - \sum_{r=1}^{k+1} RA_r) * (t_M - t'_M - t_m)$ units. Note that, the effect of this decision will influence the total flow time, but not affect the future decisions. The net decrease resulting from this decision, ND_k, can be expressed as,

$$ND_k = (n - \sum_{r=1}^{k+1} RA_r) * (t_M - t'_M - t_m) - RA_{k+1} * t'_M \quad (6.1)$$

As the decisions are independent, the optimal decision after the minor maintenance succeeding batch k is found by evaluating ND_k as follows:

- If $ND_k < 0$, then favor A1, i.e., perform the major maintenance T time units later, i.e., after completing batch $k+1$.
- If $ND_k > 0$, then favor A2, i.e., perform the major maintenance before starting batch $k+1$.
- If $ND_k = 0$, then A1 and A2 are indifferent alternatives.

6.2 Non-resumable Jobs

For non-resumable jobs case, when the minor and major maintenances are pre-planned, the problem reduces to our 1|nr-a, multi, fix|$\sum C_j$ problem for which the approach proposed in Chapter 4 can be used.

Similar to the resumable jobs case, given a sequence we can find an optimal maintenance plan after each minor maintenance. Below is the detailed description of our optimal maintenance plan for a given sequence. Note that the plan is very similar to that of resumable case.

For a given sequence, we let NA_k denote the number of jobs that are completed in batch. Note that, NA_k s are independent of the maintenance plan employed.

Assume a minor maintenance is performed succeeding batch k. If A2 is chosen, each job in batch $k+1$ will start t'_M time units later causing an increase of $NA_{k+1} * t'_M$ units in the total flow time, over A1. On the other hand, each job to be processed in batches $k+2, k+3, ..., q$ starts $(t_M - t'_M - t_m)$ time units earlier causing a decrease of $(n - \sum_{r=1}^{k+1} NA_r) * (t_M - t'_M - t_m)$ units in the total flow time. As in resumable case, the effect of this decision will influence the total flow time, but not the future decisions. Then, the net decrease resulting from this decision, ND_k is,

$$ND_k = (n - \sum_{r=1}^{k+1} NA_r) * (t_M - t'_M - t_m) - NA_{k+1} * t'_M \quad (6.2)$$

As the decisions are independent, the optimal decision after the minor maintenance succeeding batch k is found by evaluating ND_k as follows:

- If $ND_k < 0$, then favor A1, i.e., perform the major maintenance T time units later, i.e., after completing batch $k+1$.
- If $ND_k > 0$, then favor A2, i.e., perform the major maintenance before starting batch $k+1$.
- If $ND_k = 0$, then A1 and A2 are indifferent alternatives.

CHAPTER 7

CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

Recognizing the practical importance of pre-planned unavailability periods such as preventive maintenance activities, the researchers have begun to study on the machine scheduling problems with availability constraints since early nineties. Although there have been various studies in this field especially in recent years, there are still too many unexplored problems.

In this research, we address the single machine total flow time problem where the jobs are non-resumable and the machine is subject to preventive maintenance activities of known starting times and durations.

To the best of our knowledge, there is no reported study on our problem. But there are studies considering its special cases with the periodic maintenances, and single maintenance.

We formulated the problem as a mixed integer linear program and propose some properties of an optimal solution. We incorporate these properties to our branch and bound algorithm. We further enhance the efficiency of the algorithm by lower and upper bounds.

Our first upper bounding procedure, Improved SPT (ISPT), improves the SPT sequence by pairwise interchanges between two jobs in different batches. Our second upper bounding procedure, Modified SPT (MSPT), is a construction type heuristic that uses the SPT rule in forming the batches and violates SPT en route to reducing the idle time of a batch. Our computational results show that MSPT performs better than ISPT in most of the problem instances.

Our first lower bound is found by the Preemptive SPT rule which is the optimal solution of the relaxed problem with resumable jobs. Our second lower bound uses upper bounds on the number of jobs that can be processed in each batch, and assigns the jobs to the batches based on these upper bounds. We observe from our computational results that our second lower bound performs better than our first lower bound in most of the problem instances.

In our experiments, we use three different preventive maintenance patterns: periodic identical maintenance tasks, increasing maintenance durations and decreasing period lengths. We also consider random pattern which may well represent to non-maintenance unavailability periods.

The results of our computational experience reveal that the most effective mechanism on the efficiency of the branch and bound algorithm is the lower bound. We observe that excluding the lower bounds increases the computation time drastically for each problem combination. We also observe that upper bounds and reduction mechanisms improve the performance of our branch and bound algorithm, however not as significant as lower bounds.

We observe from our experiment that the length of the maintenance periods has a notable effect on the performance of our algorithm. The wider the periods are, the easier the problem is. On the other hand, we see that the effect of the maintenance duration on our algorithm's performance is not very significant. Other than maintenance related parameters, variance of the job processing times also affects the algorithm performance. When the variance of the job processing times increases, the performance of our algorithm deteriorates. Hence, the most difficult problems are the ones with short maintenance periods and high variance of job processing times. For this problem combination, our algorithm can solve problem instances with up to 35 jobs within a reasonable computation time. On the other hand, the easiest problems are the ones with long maintenance periods and low variance of job processing times and our algorithm can solve instances with up to 80 jobs of this problem combination.

We observe from our computational results that the number of nodes explored till the optimal solution is very small compared to the total number of nodes, hence most of the computational effort is spent on proving the optimality of the solution. We can conclude that, for large size problems we can design a

truncated branch and bound algorithm that terminates after evaluating a certain number of nodes or after a pre-specified CPU time. Such an algorithm can return very satisfactory solutions as the optimal solutions are likely to be obtained at early nodes.

We can extend our branch and bound algorithm to an a -approximation algorithm by inflating the lower bounds with the rate of $(1+a)$. In other words, we eliminate a partial schedule when $(1+a)LB$ is no less than the best known upper bound. An a -approximation algorithm returns good solutions with guaranteed accuracy in reasonable time, especially for the problems with longer maintenance periods as the lower bounds are very close to optimal solutions for those problem combinations.

We observe that our upper bounding procedures perform satisfactorily well. Using these procedures as starting solutions of meta-heuristic algorithms, such as tabu search or evolutionary algorithms, may be a promising future research. Moreover, one can design a heuristic procedure that is based on matching idea used in our second lower bound.

In this study we assume that maintenance start times and durations are known in advance. We only consider a particular case where the maintenance plan is selected among two alternatives. However, in practical implications, there might be cases where the maintenance start times and/or the maintenance period lengths have to be decided together with the job sequence instead of making these decisions sequentially. Hence, considering the simultaneous determination of maintenance plan and job sequence, and adapting our approach to that type of decision problem is another item of our future research list.

We assume that the jobs are non-resumable. Future research may consider semi-resumable jobs which would well-fit to practice.

Another research issue might be extending our work to the multi-machine and multi stage environments. An immediate extension might be adapting our bounding procedures to parallel machine problems. For example, our lower bounds can be adapted by allowing the job splitting together with resumability.

We assume that the preventive maintenances are sufficient to keep the system operating without any failures. Thus, we assume that no corrective

maintenance activities are necessary. However, the failures may occur in manufacturing environments, as a function of the preventive maintenance activities. Hence, an interesting future research direction might be rescheduling the jobs in a manufacturing environment where preventive and corrective maintenance activities take place simultaneously.

We study the 1|nr-a, multi, fix|$\sum C_j$ problem which resides 1|nr-a, single, fix|$\sum C_j$ as a special case. Recognizing the special structure inherent in the single maintenance case, more powerful bounding and reduction schemes can be developed.

Finally, we can study the maintenance problems in multi-criteria context. A natural multi-criteria problem arises when the minimum number of batches criterion is incorporated to the minimum total flow time criterion. In doing so, both hierarchical and simultaneous optimization problems can be considered. For example, minimizing total flow time subject to minimum number of batches is a worth-studying multi-criteria problem that primarily considers the completion time of the job set, i.e., makespan.

REFERENCES

Adiri, I., Bruno, J., Frostig, E., and Rinnooy Kan, A.H.G., 1989. "Single-machine flow-time scheduling with a single breakdown", *Acta Informatica*, Vol. 26, pp. 679-685.

Aggoune, R., 2004. "Minimizing the makespan for the flow shop scheduling problem with availability constraints", *European Journal of Operational Research*, Vol. 153, pp. 534-543.

Aggoune, R., and Portmann, M.C., 2006. "Flow shop scheduling problem with limited machine availability: A heuristic approach", *International Journal of Production Economics*, to appear.

Aktürk, M.S., Ghosh, J.B., and Güneş, E.D., 2003. "Scheduling with tool changes to minimize total completion time: A study of heuristics and their performance", *Naval Research Logistics*, Vol. 50, pp. 15-30.

Aktürk, M.S., Ghosh, J.B., and Güneş, E.D., 2004. "Scheduling with tool changes to minimize total completion time: Basic results and SPT performance", *European Journal of Operational Research*, Vol. 157, pp. 784-790.

Allaoui, H., and Artiba, A., 2006. "Scheduling two-stage hybrid flow shop with availability constraints", *Computers & Operations Research*, to appear.

Allaoui, H., Artiba, A., Elmaghraby, S.E., and Riane, F., 2006. "Scheduling of a two-machine flowshop with availability constraints on the first machine", *International Journal of Production Economics*, to appear.

Blazewicz, J., Breit, J., Formanowicz, P., Kubiak, W., and Schmidt, G., 2001. "Heuristic algorithms for the two-machine flowshop problem with limited machine availability", *Omega*, Vol. 29, pp. 599-608.

Breit, J., 2004. "An improved approximation algorithm for two-machine flow shop scheduling with an availability constraint", *Information Processing Letters*, Vol. 90, pp. 273-278.

Breit, J., 2006. "A polynomial –time approximation scheme for the two machine flowshop scheduling problem with an availability constraint", *Computers & Operations Research*, to appear.

Breit, J., Schmidt, G., and Strusevich, V.A., 2001. "Two-machine open shop scheduling with an availability constraint", *Operations Research Letters*, Vol. 29, pp. 65-77.

Breit, J., Schmidt, G., and Strusevich, V.A., 2003. "Non-preemptive two-machine open shop scheduling with non-availability constraints", *Mathematical Methods of Operations Research*, Vol. 57, pp. 217-234.

Chang, S.Y., and Hwang, H.C., 1999. “The worst-case analysis of the MULTIFIT algorithm for scheduling nonsimultaneous parallel machines”, *Discrete Applied Mathematics*, Vol. 92, pp. 135-147.

Chen, J.S., 2006a. “Single-machine scheduling with flexible and periodic maintenance”, *Journal of the Operational Research Society*, Vol. 57, pp. 703-710.

Chen, J.S., 2006b. “Optimization models for the machine scheduling problem with a single flexible maintenance activity”, *Engineering Optimization*, Vol. 38, pp. 53-71.

Chen, W.J., 2006c. “Minimizing total flow time in the single-machine scheduling problem with periodic maintenance”, *Journal of the Operational Research Society*, Vol. 57, pp. 410–415.

Cheng, T.C.E., and Liu, Z.H., 2003a. “Approximability of two-machine no-wait flowshop scheduling with availability constraints”, *Operations Research Letters*, Vol. 31, pp. 319-322.

Cheng, T.C.E., and Liu, Z.H., 2003b. “3/2-approximation for two-machine no-wait flowshop scheduling with availability constraints”, *Information Processing Letters*, Vol. 88, pp. 161-165.

Cheng, T.E.C., and Wang, G.Q., 1999. “Two-machine flowshop scheduling with consecutive availability constraints”, *Information Processing Letters*, Vol. 71, pp. 49-54.

Cheng, T.E.C., and Wang, G.Q., 2000. “An improved heuristic for two-machine flowshop scheduling with an availability constraint”, *Operations Research Letters*, Vol. 26, pp. 223-229.

Dekker, R., 1996. “Applications of maintenance optimization models: a review and analysis”, *Reliability Engineering and System Safety*, Vol. 51, pp. 229-249.

Espinouse, M.L., Formanowicz P., and Penz, B., 1999. “Minimizing the makespan in the two-machine no-wait flow-shop with limited machine availability”, *Computers & Industrial Engineering*, Vol. 137, pp. 497-500.

Gharbi, A., and Haouari, M., 2005. “Optimal parallel machines scheduling with availability constraints”, *Discrete Applied Mathematics*, Vol. 148, pp. 63-87.

Graves, G.H., and Lee, C.Y., 1999. “Scheduling maintenance and semiresumable jobs on a single machine”, *Naval Research Logistics*, Vol. 46, pp. 845-863.

Hwang, H.C., and Chang, S.Y., 1998. “Parallel machines scheduling with machine shutdowns”, *Computers & Mathematics with Applications*, Vol. 36, pp. 21-31.

Hwang, H.C., Lee, K., and Chang, S.Y., 2005. “The effect of machine availability on the worst-case performance of LPT”, *Discrete Applied Mathematics*, Vol. 148, pp. 49-61.

Kacem, I., Chu, C., and Souissi, A., 2006. “Single-machine scheduling with an availability constraint to minimize the weighted sum of the completion times”, *Computers & Operations Research*, to appear.

Kaspi, M., and Montreuil, B., 1988. “On the scheduling of identical parallel processors with arbitrary initial processor available time”, Research Report, School of Industrial Engineering, Purdue University, West Lafayette, IN 47907.

Kubiak, W., Blazewicz, J., Formanowicz, P., Breit, J., and Schmidt, G., 2002. “Two-machine flow shops with limited machine availability”, *European Journal of Operational Research*, Vol. 136, pp. 528-540.

Kubzin, M.A., and Strusevich, V.A., 2004. “Two-machine flow shop no-wait scheduling with a nonavailability interval”, *Naval Research Logistics*, Vol. 51, pp. 613-631.

Kubzin, M.A., Strusevich, V.A., and Breit, J., 2006. “Polynomial-time approximation schemes for two-machine open shop scheduling with nonavailability constraints”, *Naval Research Logistics*, Vol. 53, pp. 16-23.

Lee, C.Y., 1991. “Parallel machines scheduling with nonsimultaneous machine available time”, *Discrete Applied Mathematics*, Vol. 30, pp. 53-61.

Lee, C.Y., 1996. “Machine scheduling with an availability constraint”, *Journal of Global Optimization*, Vol. 9, pp. 395-416.

Lee, C.Y., 1997. “Minimizing the makespan in the two-machine flowshop scheduling problem with an availability constraint”, *Operations Research Letters*, Vol. 20, pp. 129-139.

Lee, C.Y., 1999. “Two-machine flowshop scheduling with availability constraints”, *European Journal of Operational Research*, Vol. 114, pp. 420-429.

Lee, C.Y., and Chen, Z.L., 2000. “Scheduling jobs and maintenance activities on parallel machines”, *Naval Research Logistics*, Vol. 47, pp. 145-165.

Lee, C.Y., He, Y., and Tang, G., 2000. “A note on "parallel machine scheduling with non-simultaneous machine available time"”, *Discrete Applied Mathematics*, Vol. 100, pp. 133-135.

Lee, C.Y., Lei, L., and Pinedo, M., 1997. “Current trends in deterministic scheduling”, *Annals of Operations Research*, Vol. 7, pp. 1-41.

Lee, C.Y., and Liman, S.D., 1992. “Single-machine flow-time scheduling with scheduled maintenance”, *Acta Informatica*, Vol. 29, pp. 375-382.

Lee, C.Y., and Liman, S.D., 1993. “Capacitated two-parallel machines scheduling to minimize sum of job completion times”, *Discrete Applied Mathematics*, Vol. 41, pp. 211-222.

Liao, C.J., and Chen, W.J., 2003. “Single-machine scheduling with periodic maintenance and nonresumable jobs”, *Computers & Operations Research*, Vol. 30, pp. 1335-1347.

Liao, C.J., Shyur, D.L., and Lin, C.H., 2005. “Makespan minimization for two parallel machines with an availability constraint”, *European Journal of Operational Research*, Vol. 160, pp. 445-456.

Lin, G.H., 1998. “The exact bound of Lee's MLPT”, *Discrete Applied Mathematics*, Vol. 85, pp. 251-254.

Mauguiere, P.H., Billaut, J.C., and Bouquard, J.L., 2005. “New single machine and job-shop scheduling problems with availability constraints”, *Journal of Scheduling*, Vol. 8, pp. 211-231.

Mosheiov, G., 1994. “Minimizing the sum of job completion times on capacitated parallel machines”, *Mathematical and Computer Modelling*, Vol. 20, pp. 91-99.

Newbrough, E. T., 1967. *Effective Maintenance Management*. McGraw-Hill.

Ng, C.T., and Kovalyov, M.Y., 2004. “An FPTAS for scheduling a two-machine flowshop with one unavailability interval”, *Naval Research Logistics*, Vol.51 pp. 307-315.

Pintelon, L.M., and Gelders, L.F., 1992. “Maintenance management decision making ”, *European Journal of Operational Research*, Vol. 58, pp. 301-317.

Pinedo, M., 1995. *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, Inc.

Qi, X., Chen, T., and Tu, F., 1999. “Scheduling the maintenance on a single machine”, *Journal of the Operational Research Society*, Vol. 50, pp. 1071-1078.

Sadfi, C., Penz, B., Rapine, C., Blazewicz, J., and Formanowicz, P., 2005. “An improved approximation algorithm for the single machine total completion time scheduling problem with availability constraints”, *European Journal of Operational Research*, Vol. 161, pp. 3-10.

Sanlaville, E., and Schmidt, G., 1998. “Machine scheduling with availability constraints”, *Acta Informatica*, Vol. 35, pp. 795-811.

Schmidt, G., 2000. “Scheduling with limited machine availability”, *European Journal of Operational Research*, Vol. 121, pp. 1-15.

Simeu-Abazi, Z., and Sassine, C., 2001. “Maintenance integration in manufacturing systems: from the modeling tool to evaluation”, *The International Journal of Flexible Manufacturing Systems*, Vol. 13, pp. 267-285.

Smith, W.E., 1956. “Various optimizers for single stage production”, *Naval Research Logistics*, Vol. 3, pp. 59-66.

Suresh, V., and Chaudhuri, D., 1996. “Scheduling of unrelated parallel machines when machine availability is specified”, *Production Planning & Control*, Vol. 7, pp. 393-400.

Wang, G.Q., and Cheng, T.C.E., 2001. “Heuristics for two-machine no-wait flowshop scheduling with an availability constraint”, *Information Processing Letters*, Vol. 80, pp. 305-309.

Wang, G.Q., Sun, H.Y., and Chu, C.B., 2005. “Preemptive scheduling with availability constraints to minimize total weighted completion times”, *Annals Of Operations Research*, Vol. 133, pp. 183-192.

Worsham, W.C., 2000. “Is preventive maintenance necessary?”, *Maintenance Resources On-Line Magazine*, Sep/Oct 2000.

Appendix 4.1 Illustration of ISPT Heuristic with a Numerical Example

j	1	2	3	4	5	6	7	8	9	10
p_j	2	3	3	4	5	6	7	9	10	10

$n = 10$, $T = 20$ and $t = 4$

ISPT

Construction:

Step 0. $j = 1, k = 1, I_1 = T_1 = 20$, *Step 1.* $p_1 = 2 \le I_1 = 20,\ C_1 = 2,\ I_1 = I_1 - p_1 = 18$

Step 2. $j = 2$, *Step 1.* $p_2 = 3 \le I_1 = 18,\ C_2 = C_1 + p_2 = 5,\ I_1 = I_1 - p_2 = 15$

Step 2. $j = 3$, *Step 1.* $p_3 = 3 \le I_1 = 15,\ C_3 = C_2 + p_3 = 8,\ I_1 = I_1 - p_3 = 12$

Step 2. $j = 4$, *Step 1.* $p_4 = 4 \le I_1 = 12,\ C_4 = C_3 + p_4 = 12,\ I_1 = I_1 - p_4 = 8$

Step 2. $j = 5$, *Step 1.* $p_5 = 5 \le I_1 = 8,\ C_5 = C_4 + p_5 = 17,\ I_1 = I_1 - p_5 = 3$

Step 2. $j = 6$, *Step 1.* $p_6 = 6 > I_1 = 3$, it is not feasible to place job 6 in batch 1.

Step 3. $k = 2, I_2 = T_2 = 20$,

Step 1. $p_6 = 6 \le I_2 = 20,\ C_6 = T_1 + t_1 + p_6 = 30,\ I_2 = I_2 - p_6 = 14$

Step 2. $j = 7$, *Step 1.* $p_7 = 7 \le I_2 = 14,\ C_7 = C_6 + p_7 = 37,\ I_2 = I_2 - p_7 = 7$

Step 2. $j = 8$, *Step 1.* $p_9 = 9 > I_2 = 7$, it is not feasible to place job 8 in batch 2.

Step 3. $k = 3, I_3 = T_3 = 20$,

Step 1. $p_8 = 9 \le I_3 = 20,\ C_9 = \sum_{r=1}^{2}(T_r + t_r) + p_8 = 57,\ I_3 = I_3 - p_8 = 11$

Step 2. $j = 9$, *Step 1.* $p_9 = 10 \le I_3 = 11,\ C_9 = C_8 + p_9 = 67,\ I_3 = I_3 - p_9 = 1$

Step 2. $j = 10$, *Step 1.* $p_{10} = 9 > I_3 = 1$, it is not feasible to place job 9 in batch 3.

Step 3. $k = 4, I_4 = T_4 = 20$,

Step 1. $p_{10} = 10 \le I_4 = 20,\ C_{10} = \sum_{r=1}^{3}(T_r + t_r) + p_{10} = 82,\ I_4 = I_4 - p_{10} = 10$

Step 2. All jobs are sequenced. There are $q = 4$ batches in S and the total flow time of S is $F = \sum_{j=1}^{10} C_j = 317$.

S can be represented by

$\{2, 3, 3, 4, 5, I_1 = 3, t_1 = 4; 6, 7, I_2 = 7, t_2 = 4; 9, 10, I_3 = 1, t_3 = 4; 10\}$.

In this representation, processing time of the sequenced jobs, idle time of each batch and duration of each maintenance task is given in the order of taking place in time.

Improvement:

Step 0. Let $k=1$.

Step 1. $k<q=4$

Step 2. $I_1=3>0$

Step 3. $j=$ the last job of batch 1=5

Step 4. $r=k+1=2$

Step 5. $r\le q$, $i=1$

Step 6. $p_{[51]}=5, p_{[12]}=6, I_1=3, I_2=7$. Since $p_{[51]}<p_{[12]}$ and $I_1+p_{[51]}\ge p_{[12]}$, jobs [51] and [12] can be interchanged feasibly. In the total flow time, the coefficient of $p_{[51]}$ is 1 as it is the last job in batch 1 and the coefficient of $p_{[12]}$ is 2 since it is the first job in batch 2 that has 2 jobs. So, interchanging jobs [51] and [12] reduces the total flow time. As the two conditions of Step 6 is satisfied, these two jobs are interchanged and I_1 and I_2 are updated:

$I_1=I_1+p_{[12]}-p_{[51]}=3+5-6=2$, $I_2=I_2+p_{[51]}-p_{[12]}=7+6-5=8$.

After this interchange, S becomes

$\{2,3,3,4,6,I_1=2,t_1=4;5,7,I_2=8,t_2=4;9,10,I_3=1,t_3=4;10\}$

Step 7. $i=i+1=2$

Step 6. $p_{[51]}=6, p_{[22]}=7, I_1=2, I_2=8$. Since $p_{[51]}<p_{[22]}$ and $I_1+p_{[51]}\ge p_{[22]}$, jobs [51] and [22] can be interchanged feasibly. In the total flow time, the coefficient of both $p_{[51]}$ and the coefficient of $p_{[22]}$ is 1. So, interchanging these two jobs does not increase the total flow time. As the two conditions of Step 6 is satisfied, these two jobs are interchanged and I_1 and I_2 are updated:

$I_1=I_1+p_{[22]}-p_{[51]}=1$, $I_2=I_2+p_{[51]}-p_{[22]}=9$.

After this interchange, S becomes

$\{2,3,3,4,7,I_1=1,t_1=4;5,6,I_2=9,t_2=4;9,10,I_3=1,t_3=4;10\}$

Step 7. $i=i+1=3>n_2$, $r=r+1=3$.

Step 5. $r \le q,\ i = 1$

Step 6. $p_{[51]} = 7, p_{[13]} = 9, I_1 = 1, I_3 = 1$. Since $I_1 + p_{[51]} < p_{[13]}$, interchanging the jobs [51] and [13] is not feasible. As processing times of all jobs after [13] are greater than $p_{[13]}$, there is no other job that can feasibly swapped with job [51].

Step 8. $j = j - 1 = 4,\ r = k + 1 = 2$

Step 5. $r \le q,\ i = 1$

Step 6. $p_{[41]} = 4, p_{[12]} = 5, I_1 = 1, I_2 = 9$. Since $p_{[41]} < p_{[12]}$ and $I_1 + p_{[41]} \ge p_{[12]}$, jobs [41] and [12] can be interchanged feasibly. In the total flow time, the coefficient of both $p_{[51]}$ and the coefficient of $p_{[22]}$ is 2. So, interchanging these two jobs does not increase the total flow time. As the two conditions of Step 6 is satisfied these two jobs are interchanged and I_1 and I_2 are updated:

$I_1 = I_1 + p_{[22]} - p_{[41]} = 0,\ I_2 = I_2 + p_{[41]} - p_{[12]} = 10$.

After this interchange, S becomes

$\{2, 3, 3, 5, 7, I_1 = 0, t_1 = 4; 4, 6, I_2 = 10, t_2 = 4; 9, 10, I_3 = 1, t_3 = 4; 10\}$

Since $I_1 = 0, k = k + 1 = 2$.

Step 1. $k < q = 4$

Step 2. $I_2 = 10 > 0$

Step 3. $j =$ the last job of batch 2=2

Step 4. $r = k + 1 = 3$

Step 5. $r \le q,\ i = 1$

Step 6. $p_{[22]} = 6, p_{[13]} = 9, I_2 = 10, I_3 = 1$. Since $p_{[22]} < p_{[13]}$ and $I_2 + p_{[22]} \ge p_{[13]}$, jobs [22] and [13] can be interchanged feasibly. In the total flow time, the coefficient of $p_{[22]}$ is 1 and the coefficient of $p_{[13]}$ is 2. So, interchanging these two jobs reduces the total flow time. As the two conditions of Step 6 is satisfied, these two jobs are interchanged and I_2 and I_3 are updated:

$I_2 = I_2 + p_{[13]} - p_{[22]} = 7,\ I_3 = I_3 + p_{[22]} - p_{[13]} = 4$.

After this interchange, S becomes

$\{2, 3, 3, 5, 7, I_1 = 0, t_1 = 4; 4, 9, I_2 = 7, t_2 = 4; 6, 10, I_3 = 4, t_3 = 4; 10\}$

Step 7. $i = i + 1 = 2$

Step 6. $p_{[22]} = 9, p_{[23]} = 10, I_2 = 7, I_3 = 4$. Since $p_{[22]} < p_{[23]}$ and $I_2 + p_{[22]} \geq p_{[23]}$, jobs [22] and [23] can be interchanged feasibly. In the total flow time, the coefficient of both $p_{[22]}$ and $p_{[13]}$ is 1. So, interchanging these two jobs does not increase the total flow time. As the two conditions of Step 6 is satisfied, these two jobs are interchanged and I_2 and I_3 are updated:

$I_2 = I_2 + p_{[23]} - p_{[22]} = 6$, $I_3 = I_3 + p_{[22]} - p_{[23]} = 5$.

After this interchange, S becomes

$\{2,3,3,5,7, I_1 = 0, t_1 = 4; 4,10, I_2 = 6, t_2 = 4; 6,9, I_3 = 5, t_3 = 4; 10\}$

Step 7. $i = i + 1 = 3 > n_3$ as there are 2 jobs in batch 3. $r = r + 1 = 4$

Step 5. $i = 1$

Step 6. $p_{[22]} = p_{[14]} = 10$. Since these two jobs are identical, do not interchange them.

Step 7. $i = i + 1 = 2 > n_4$ as there is only 1 job in batch 4. $r = r + 1 = 5$

Step 5. $r > q$

Step 8. $j = j - 1 = 1$, $r = k + 1 = 3$

Step 5. $i = 1$

Step 6. $p_{[12]} = 4, p_{[13]} = 6, I_2 = 6, I_3 = 5$. Since $p_{[12]} < p_{[13]}$ and $I_2 + p_{[12]} \geq p_{[13]}$, jobs [12] and [13] can be interchanged feasibly. In the total flow time, the coefficient of both $p_{[12]}$ and $p_{[13]}$ is 2. So, interchanging these two jobs does not increase the total flow time. As the two conditions of Step 6 is satisfied, these two jobs are interchanged and I_2 and I_3 are updated:

$I_2 = I_2 + p_{[13]} - p_{[12]} = 4$, $I_3 = I_3 + p_{[12]} - p_{[13]} = 7$.

After this interchange, S becomes

$\{2,3,3,5,7, I_1 = 0, t_1 = 4; 6,10, I_2 = 4, t_2 = 4; 4,9, I_3 = 7, t_3 = 4; 10\}$

Step 7. $i = i + 1 = 2$

Step 6. $p_{[12]} = 6, p_{[23]} = 9, I_2 = 4, I_3 = 7$. Since $p_{[12]} < p_{[23]}$ and $I_2 + p_{[12]} \geq p_{[23]}$, jobs [12] and [23] can be interchanged feasibly. But in the total flow time, the coefficient of $p_{[12]}$ is 2 and $p_{[23]}$ is 1. So, these two jobs are not interchanged not to increase the total flow time.

Step 7. $i = i + 1 = 3 > n_3$, $r = r + 1 = 4$

Step 5. $r \le q, i = 1$

Step 6. $p_{[12]} = 6, p_{[14]} = 10, I_2 = 4$. Since $p_{[12]} < p_{[14]}$ and $I_2 + p_{[12]} \ge p_{[14]}$, jobs [12] and [14] can be interchanged feasibly. But in the total flow time, the coefficient of $p_{[12]}$ is 2 and $p_{[14]}$ is 1. So, these two jobs are not interchanged not to increase the total flow time.

Step 7. $i = i + 1 = 2 > n_4$, $r = r + 1 = 5$

Step 5. $r > q$

Step 8. $j = j - 1 = 0$, $k = k + 1 = 3$

Step 1. $k < q$

Step 2. $I_3 = 7 > 0$

Step 3. $j =$ the last job of batch $3 = 2$.

Step 4. $r = k + 1 = 4$

Step 5. $i = 1$

Step 6. $p_{[23]} = 9, p_{[14]} = 10, I_3 = 7$. Since $p_{[23]} < p_{[14]}$ and $I_2 + p_{[23]} \ge p_{[14]}$, jobs [23] and [14] can be interchanged feasibly. In the total flow time, the coefficient of both $p_{[23]}$ and $p_{[14]}$ is 1. So, interchanging these two jobs does not increase the total flow time. As the two conditions of Step 6 is satisfied, these two jobs are interchanged and I_3 is updated:

$I_3 = I_3 + p_{[14]} - p_{[23]} = 6$.

After this interchange, S becomes

$\{2, 3, 3, 5, 7, I_1 = 0, t_1 = 4; 6, 10, I_2 = 4, t_2 = 4; 4, 10, I_3 = 6, t_3 = 4; 9\}$

Step 7. $i = i + 1 = 2 > n_4$, $r = r + 1 = 5$

Step 5. $r > q$

Step 8. $j = j - 1 = 1$, $r = k + 1 = 4$

Step 5. $r \le q, i = 1$

Step 6. $p_{[13]} = 4, p_{[14]} = 9, I_3 = 6$. Since $p_{[13]} < p_{[14]}$ and $I_2 + p_{[13]} \ge p_{[14]}$, jobs [13] and [14] can be interchanged feasibly. But in the total flow time, the coefficient of $p_{[13]}$

is 2 and $p_{[14]}$ is 1. So, these two jobs are not interchanged not to increase the total flow time.

Step 7. $i = i + 1 = 2 > n_4$, $r = r + 1 = 5$

Step 5. $r > q$

Step 8. $j = j - 1 = 0$, $k = k + 1 = 4$

Step 5. $k = q$, stop.

$S = \{2,3,3,5,7, I_1 = 0, t_1 = 4; 6,10, I_2 = 4, t_2 = 4; 4,10, I_3 = 6, t_3 = 4; 9\}$ is the schedule obtained by ISPT and the corresponding total flow time is

$$F = \sum_{j=1}^{10} C_j = 2 + 5 + 8 + 13 + 20 + 30 + 40 + 52 + 62 + 81 = 313 .$$

Appendix 4.2 Illustration of MSPT Heuristic with a Numerical Example

j	1	2	3	4	5	6	7	8	9	10
p_j	2	3	3	4	5	6	7	9	10	10

$n = 10$, $T = 20$ and $t = 4$

MSPT

Step 0. $k = 1,\ S_1 = \varnothing,\ US = \{1,2,3,4,5,6,7,8,9,10\}$.

Step 1. First five jobs are assigned to batch 1 and the resulting slack time is $I_1 = 3$.

$S_1 = \{1,2,3,4,5\},\ US = \{6,7,8,9,10\}$.

Step 2. $I_1 = 3 > 0$.

$p_{[51]} + I_1 \geq p_6,\ j = 1$.

Step 3. $j = 1 \leq n_1 = 5$.

Step 4. $p_{[11]} + I_1 < p_6$. First job of batch 1 can not be replaced by any of the unscheduled jobs. $j = j + 1 = 2$.

Step 3. $j = 2 \leq n_1 = 5$

Step 4. $p_{[21]} + I_1 \geq p_6$. Second job of batch 1 can be feasibly replaced by one of the unscheduled jobs.

$p_6 = \max_{h \in US}\{p_h \mid p_h \leq p_{[21]} + I_1\}$, so replace job [21] with job 6.

$I_1 = I_1 + p_2 - p_6 = 0,\ S_1 = \{1,6,3,4,5\},\ US = \{2,7,8,9,10\}$.

Step 5. $I_1 = 0$.

Step 6. The jobs in batch 1 are reordered in the SPT order. $S_1 = \{1,3,4,5,6\}$.

$k = k + 1 = 2$

Step 1. The first three unscheduled jobs, i.e., jobs 2, 7 and 8, are assigned to batch 2 and the resulting slack time is $I_2 = 1$.

$S_2 = \{2,7,8\},\ US = \{9,10\}$.

Step 2. $I_2 = 1 > 0$.

$p_{[32]} + I_2 \geq p_9,\ j = 1$.

Step 3. $j = 1 \leq n_2 = 3$.

Step 4. $p_{[12]} + I_2 < p_9$. First job of batch 2 can not be replaced by any of the unscheduled jobs. $j = j+1 = 2$.

Step 3. $j = 2 \le n_2 = 3$

Step 4. $p_{[22]} + I_2 < p_9$. Second job of batch 2 can not be replaced by any of the unscheduled jobs. $j = j+1 = 3$.

Step 3. $j = 3 \le n_2 = 3$

Step 4. $p_{[32]} + I_2 \ge p_9$. Third job of batch 2 can be feasibly replaced by one of the unscheduled jobs.

$p_{10} = \max_{h \in US}\{p_h \mid p_h \le p_{[32]} + I_2\}$, so replace job [32] with job 10.

$I_2 = I_2 + p_8 - p_{10} = 0,\ S_2 = \{2,7,10\},\ US = \{8,9\}$.

Step 5. $I_2 = 0$.

Step 6. The jobs in batch 2 are reordered in the SPT order. $S_2 = \{2,7,10\}$.

$k = k+1 = 3$

Step 1. All unscheduled jobs, i.e., the jobs 8 and 9, are assigned to batch 3 and the resulting slack time is $I_3 = 1$.

$S_3 = \{8,9\},\ US = \varnothing$.

$S = S_1 \cup S_2 \cup S_3 = \{1,3,4,5,6;2,7,10;8,9\}$.

S is the schedule obtained by MSPT and the corresponding total flow time is $F = \sum_{j=1}^{10} C_j = 2+5+9+14+20+27+34+44+57+67 = 279$. The final schedule S is illustrated below