MAKESPAN MINIMIZATION FOR PARALLEL MACHINES SCHEDULING WITH AVAILABILITY CONSTRAINTS

by

Navid Hashemian

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF APPLIED SCIENCE

Major Subject: Industrial Engineering

at

DALHOUSIE UNIVERSITY

Halifax, Nova Scotia March, 2010

Dalhousie University

Faculty of Engineering

Department of Industrial Engineering

The undersigned hereby certify that they have examined, and recommend to the Faculty of Graduate Studies for acceptance, the thesis entitled "**Makespan Minimization for Parallel Machines Scheduling with Availability Constraints**" by **Navid Hashemian** in partial fulfillment of the requirements for the degree of **Master of Applied Science**.

Dated: _____

Supervisor: _____
Claver Diallo

Examiners: _____
Abdul-Rahim Ahmad

_____
Uday Venkatadri

_____
Eldon Gunn

Dalhousie University

Faculty of Engineering

DATE: _____

AUTHOR:	Navid Hashemian
TITLE:	**Makespan Minimization for Parallel Machines Scheduling with Availability Constraints**
MAJOR SUBJECT:	Industrial Engineering
DEGREE:	Master of Applied Science
CONVOCATION:	May, 2010

to

Dr. Béla Vizvári

for continuing to believe in me

Table of Contents

List of Tables

List of Figures

List of Abbreviations

CPUt	Central Processing Unit time
ILP	Integer Linear Programming
LPT	Longest Processing Time
LS	List Scheduling
MLPT	Modified Longest Processing Time
NP-hard	Non-deterministic Polynomial-time hard
SPT	Shortest Processing Time

Abstract

A new method is developed to schedule jobs on parallel machines with availability constraints. The objective of the problem is to minimize the makespan of the total production schedule. Without the availability constraints the scheduling of machines is a $P_m \mid\mid C_{max}$ problem. The scheduling of this problem was the topic of many earlier papers.

The main contribution of this research is that the schedule of the jobs on parallel machines with availability constraints is determined within a single implicit enumeration algorithm. Within the general enumeration scheme, the loads of each machine are enumerated in a lexicographic order. An exact integer linear programming model is provided, too. The difficulty of the problem depends on the properties of the processing times, the number of machines, and the number of availability constraints on the machines. In some subclasses, problems with very large number of jobs are solved. The largest problems solved within one hour limit have $1,000,000$ jobs.
keywords: parallel machines; availability contraints; makespan; scheduling;

Acknowledgements

First and foremost, I would like to express my gratitude to Dr. Claver Diallo for his supervision, advice, and guidance throughout my studies. Above all and the most needed, he provided me unflinching encouragement and support in various ways. Without him, this thesis would not have been possible.

My sincere thanks are due to the official referees Dr. Eldon Gunn, Dr. Uday Venkatadri and Dr. Abdul-Rahim Ahmad for their time and constructive comments on this thesis. I am thankful that in the midst of all their activities, they have accepted sit on the supervisory and examining committees. Their kind support and guidance have provided great value to this work.

I am thankful to the chair of our department, Dr. John T. Blake, for providing excellent facilities.

I would like to express my gratitude to Cindi Slaunwhite and Mary-Anne Wensley for their kind support.

I want to express my sincere thanks to Dr. Béla Vizvári for his guidance and support throughout my academic career. Without him, this thesis would not have been possible.

Last but not least, it shall be emphasized that the continuous support of my family and my friends provided especially appreciated source of motivation. I am particularly indebted to my parents.

Chapter 1

Introduction

In this chapter, we introduce basic concepts of scheduling along with an outline of the remainder of the thesis. Section 1.1 gives a basic introduction to scheduling. In Section 1.2 the objectives and the motivations of the thesis are discussed. Section 1.3 gives a brief introduction to machine scheduling.

1.1 Introduction to Scheduling

Scheduling is a very common activity in both industry and non-industry settings. Everyday, meetings are scheduled, deadlines are set for projects, vacation and work periods are set, maintenance and upgrade operations are planned, operation rooms are booked and sports games are scheduled and arenas booked, etc.

Proper scheduling allows various activities, jobs or tasks to be executed in an organized manner, while preventing resource conflicts. Example of activities are: the different stages of a research project, the tasks a nurse has to perform during a work day, the manufacturing operations in a semi-conductor company, etc. Objectives maybe to minimize the time to complete all activities, minimize the lateness of activities that cannot be completed on time, the completion of the most important activities on time, maximizing the number of customers or patients served, etc.

In general, intelligent scheduling methods are needed to assign activities to processors (machines) when faced with limited execution time and scarce resources. Many researchers have and are working on the topic, specially in the area of machine scheduling. However, most assume that the processors or machines are always available over the course of the production horizon, which is not always realistic.

1.2 Motivation and Research Goal

Classical machine scheduling problems assume that machines are continuously available over the scheduling horizon. This assumption might be justified in some cases but it is not satisfied in many practical situations. The operation of a machine can be interrupted for a certain period of time due to accidental breakdown, preventive maintenance, periodic repair or other reasons, which render the machine nonproductive for a certain period of time.

In industry, occurrences like the ones describe above are not uncommon. It is therefore necessary to find ways of scheduling jobs or tasks in the presence of non-availability periods. As the next example will show, scheduling jobs without specifically considering the non-availability constraints can lead to very poor results.

In this example, two jobs with processing times of 1 and 2 units of time are to be scheduled on 2 parallel machines M_1 and M_2. Machine M_1 is unavailable from instant 1 to instant n. Machine M_2 is available during the whole scheduling horizon. Figure 1.1 shows the schedule on these two machines obtained by the Longest Processing Time (LPT) algorithm, which assign jobs on the machines by the non-increasing order of their processing times. This algorithm is the most common heuristic in the literature for parallel machines scheduling.

According to the LPT algorithm, the first job to be scheduled is the one with duration 2. On machine 1, the non-availability period is reached before the job is completed. Therefore the processing will only resume after instant n and complete at n+1.

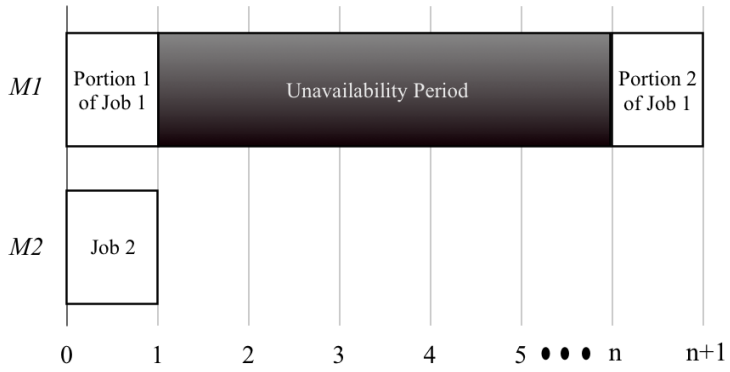

Figure 1.1: Scheduling without Considering the Availability Constraints

The second job with duration of 1 is scheduled on machine 2. The makespan of this example is n+1. The solution of this method can be very large if the unavailability period of length (n-1) is long.

Another motivation to study machine scheduling with availability constraints is the presence of special tasks. A special task is a task that should be processed within a specific time interval. Examples of special jobs include tasks with higher priority or tasks previously booked for processing. Those special jobs can be represented by the non-availability periods on the different machines. There are many of other examples where the investigation of machine scheduling with availability constraint is of great importance. Hence this topic has recently attracted attention. However, this consideration adds complexity to any scheduling problem, even in the case of two machines with a single unavailability period.

The goals of this research are to:

i) Conduct an extensive literature review on parallel machine scheduling with availability constraints.

ii) Provide a mathematical model for the problem: Since the only exact method in the literature can apply to only the two machine problem [31], it was necessary to develop new model.

iii) Develop an exact algorithm: The main result of this study is a new algorithm for the parallel machine scheduling with availability constraints. The

aim is to compare the computing time and the size of the solved problems if the exact algorithm is applied or a general problem solver is applied to the model.

1.3 Preliminaries

In the following, we introduce the notation of Graham et al. [15], extended to include availability constraint. This notation consist of three fields $\alpha \mid \beta \mid \gamma$.

The first field α represents the machine environment. In the literature, three types of environment for machines are defined [36]. However, an environment may be divided into the several other environments as listed below:

- Single Machine: there is only one machine to process the tasks or jobs.
- Parallel Machines: more than one machine is performing the same function. The machines can be:
  - Identical. all machines have the same speed factors, and they can process all the jobs.
  - Uniform. parallel machine system with different speed factor, and each job has a single operation.
  - Unrelated. there is no relation between machines.
- Dedicated Machines: Machine are specialized for the execution of certain operations.
  - Flow shop: The number of operations for each job is the same as the number of machines. The first operation starts on machine 1, the second requires processing on machine 2, and so on. All jobs visit the same machines in the same sequence.
  - Job shop: The jobs are passed through machines in a different order. In other words, each job has a given sequence of operations on the machines.

– Open shop: Machines have different speed factors, and jobs should be processed on every single machine.

The second field β denotes the job characteristics these include presence of preemption or not, how jobs are resumed, existence of non-availability periods. If preemption is permitted then the processing of a job can be interrupted in order to let another job such as a rush order with high priority be processed on the machine. The job taken off the machine is said to be preempted.

In our case (presence of non-availability period), the processing of a job may be interrupted when some of its processing time extends into an unavailability interval. After the unavailability period the interrupted job can be completed based on the type of preemption. There are several forms of preemption. One form classifies preemption as *operation preemption* or *arbitrary preemption*. Under the operation preemption, the preempted job is to be processed on the same machine it started on. Thus the preempted job, must remain on that machine until it can resume. Under arbitrary preemption, a job may be preempted and sent to another machine, or continued later. A schedule is called non-preemptive, if a job can continue its processing without interruption [36].

When a job is interrupted by a non-availability period, it resumes processing at the end of the non-availability. The duration of the interrupted job, after the non-availability period, depends on the *resumability* or *non-resumability* of jobs. A job is considered resumable when its remaining processing time can be executed after the unavailability period without any time penalty. A non-resumable job is a job that must be restarted from the beginning after the unavailability period, rather than continuing its processing [25]. Finally, Lee [26] defines a semi-resumable job as a job that must be partially restarted for the portion that has been processed.

For example consider a job with a processing time of 2 units of time, being processed on a machine with an availability constraint. The machine is not available from 1 to 2. The figure below shows how the type of this job can affect its processing time.

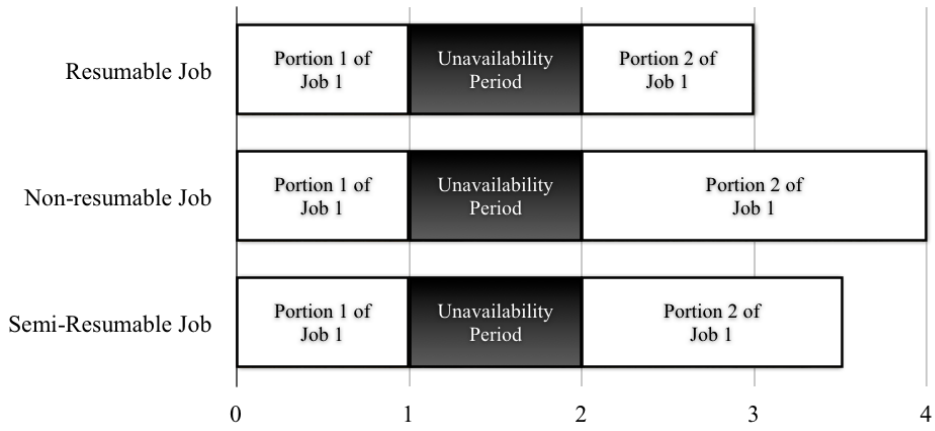

Figure 1.2: Different Job Types

The third field γ, describes the performance measure being considered or the optimality criteria. Minimizing *makespan*, *maximum lateness*, number of *tardy jobs* and *sum of completion times* are examples of performance measures.

Many different types of problem can be generated by varying each of three fields in Graham's notation. The approach that is selected to solve each of these problems is depended on the degree of information that is available for that problem. In some problems, there might be no information on the duration of the unavailability period or no clear knowledge of the start the unavailability period. A typical example is an unexpected machine breakdown. We cannot predict the accidental machine breakdown, and the scheduling should be done regardless of the data or information about the unavailability period. In some other problems, there is a partial knowledge of the availability constraints on the machines. For example, the duration of the unavailability period is known, but there is no information about the start of unavailability. For some other problems, we might have all the information on the problem, before we decide anything about the scheduling. Systematic preventive maintenance on the machines falls in the latter category. In the systematic preventive maintenance, the time and duration of the maintenance is known in advance. Schmid [42] defines three types of algorithms, based on the type of the information that one might have about a problem:

1. On-Line Algorithm: It proceeds sequentially and it only needs to know at each

instant t, the number of jobs ready at t, the number of machines available at t, the remaining processing time and their deadlines or due dates.

2. Nearly On-Line Algorithm: It needs in addition at time t the time of the next event, that is either number of unavailable machines change or a new job becomes ready for processing ([40], extended from [22]).

3. Off-Line Algorithm: It needs all the problem data in advance. All the information concerning machine availabilities, and job characteristics is determined before the start of the algorithm.

Problems can also be classified according to the pattern of the non-availability periods on the parallel machines. Schmid [41] and Liu and Sanlaville [35] define 6 different patterns in the scheduling of parallel machines: constant, zigzag, increasing (decreasing), increasing (decreasing) zigzag, staircase and arbitrary. Schmid [42] lists some algorithms developed to address the makespan problem with specific non-availability patterns. In the literature, there is no effective pattern-independent algorithm.

In this research, we consider the following scheduling problem: $P_m \mid r-a_{i,q} \mid C_{max}$. The symbol P_m in the α field denotes the parallel machine problem, r in the β field represents resumable jobs, and $a_{i,q}$ shows the number of machines with availability constraints, where q represents the maximum number of availability constraints on a machine. In the case of single machine and one availability constraint $a_{i,q}$ is equal to a. The type of preemption is assumed to be operation preemption. C_{max} in the γ field shows the optimality criterion, which is minimizing makespan, or maximum completion time. Makespan, is the time interval between the start of the first job on the machines and the completion time of the last job. This study assumes that all the information is known in advance. That is, the type of the algorithm is off-line, or deterministic.

The next chapter presents a basic introduction to the parallel machines scheduling with availability constraints. Later, it reviews earlier studies and researches done on the topic.

The rest of this study is organized as follow: In Chapter 3, the problem is defined and is modelled mathematically. The exact algorithm to be studied is presented in Chapter 4. Chapter 5 presents a computational analysis to evaluate the performance of the exact algorithm under a variety of experimental conditions. Lastly, chapter 6 presents the conclusions along with some suggestions and areas for future study.

Chapter 2

Literature Review

In this chapter, the first section gives an introduction to machine scheduling, and provides a brief overview of basic scheduling concepts. The second section describes the availability constraints in machine scheduling. Finally, the third section reviews the main studies and research conducted on this topic.

2.1 Parallel Machine Scheduling

Schedules are generally evaluated by a performance measure or an objective function. A popular performance measure is the minimization of the makespan. Makespan, or maximum completion time is the time interval between starting the first job on a machine and the completion of the last job. In the literature makespan is denoted by C_{max}.

When there is only one machine and jobs are sequence-independent, the solution of the makespan problem is trivial: any scheduling sequence results in the optimal solution. However, the same statement is not true, when the number of machines is greater than one.

Classical parallel machine scheduling considers series of identical machines with a number of jobs and different processing times. It assumes that the jobs are ready at time zero, and machines are continuously available during the whole scheduling horizon. The simplest makespan problem arises in classical parallel machine scheduling, when jobs are sequence-independent and preemption is allowed. When preemption is permitted, the processing of a job can be interrupted and the remaining processing can be completed later, perhaps on a different machine. When preemption of the jobs

is allowed on all machines, the minimum makespan is obtained by:

$$M^* = \left\lceil \sum_{j=1}^{N} p_j/m \right\rceil \tag{2.1}$$

where N is the number of jobs, p_j is the processing time of task j, and m is the number of machines. In the case of integer processing times, the solution of (2.1) is optimal, when the result is integer. The integer result shows that the total processing time is evenly allocated among all machines. If the result is not integer then the optimal solution is equal to $\lceil M^* \rceil$ or greater.

By prohibiting preemption, makespan minimization becomes a Non-deterministic Polynomial-time hard (NP-hard) problem. Leung [30] defines NP as a class of decision problems which have "succinct" certificates (certificates whose size is bounded by a polynomial function of the size of the input) that can be verified in polynomial time. According to him problem Z is NP-hard if all problems in the NP-class are reducible to Z. According to Brucker [6]not all NP-hard problems are equally hard from practical aspect. For example there are some NP-hard problems that can be solved pseudopolynomially using dynamic programming.

It is shown that parallel machine makespan-minimization problem is NP-hard even for the two-machine problem [29]. However the two-machine problem can be solved by the pseudopolynomial algorithm [3]. But solving problems with more than two machines is very challenging. Solving these problems need a general-purpose method such as dynamic programming or branch and bound algorithm. In the case of dynamic programming, the algorithm can only be applied to relatively small sized problems. This is due to the high computation and memory requirements of this method. In the case of branch and bound, it is very difficult to obtain tight lower bounds [3].

Although, it is not easy to find the optimal makespan, a local optimal solution can always be found by heuristics. A well-known heuristic in the literature is List Scheduling (LS). In this method, jobs are sorted according to a predefined order. For example, jobs can be sorted according to the non-decreasing order of their processing

time. Then, whenever a machine becomes available, a job with the lowest index among the unscheduled jobs is assigned to that machine space. The procedure continues until all jobs are assigned to the machines.

The solution of makespan in LS depends on how the jobs are ordered. Unfortunately there is no easy way to find an optimal list. The only method to find the optimal list is to check every possible order. If there are N jobs then $N!$ permutations should be checked to find the optimal solution. Graham [13] proves that applying LS to a classical parallel machine problem yields a performance guarantee of $C_{LS}/C^* \leq 2 - 1/m$. Baker and Trietsch [3] defined performance guarantee as follow:

> performance guarantee is a bound on the performance of a particular solution method. In the case of makespan problems, it is an upper bound on the suboptimality of the makespan produced by a give heuristic procedure.

In the above performance guarantee found for the LS algorithm, the right hand side of the inequality is an error bound, which represents the ratio of the heuristic solution to the optimal solution as a function of the number of machines.

The most prominent heuristic for makespan problem is the so-called Longest Processing Time (LPT). This heuristic is a LS algorithm with the list of the jobs sorted in the non-increasing order of their processing times. Graham [14] proves that the LPT algorithm has a performance guarantee of $C_{LPT}/C^* \leq 4/3 - 1/(3m)$, which provides a better error bound than the LS algorithm.

Many other heuristics have been proposed in the literature to address the classical parallel machine scheduling problem (e.g. MULTIFIT algorithm [10], and Repetitive Modified Greedy algorithm [23]).

2.2 Parallel Machine Scheduling with Availability Constraints

It is always possible that a job cannot complete its processing before the start of an unavailability period on the machine. The processing of the job is therefore interrupted until the unavailability period elapses. In general, this type of interruption is

caused by a machine breakdown, preventive maintenance, or the arrival of a special job. For machine breakdowns and special jobs with random arrivals, the unavailability periods are stochastic: the unavailability periods are decision variables and occur randomly. On-line algorithms are needed to solve this type of problems. For preventive maintenance activity or special jobs with predetermined arrival and duration, there is complete information on the duration of the unavailabilities and the problem can be solved by off-line algorithms. It is also possible that a machine is not available at the beginning of a scheduling horizon because of a job whose processing is overflowing from the previous scheduling period to the current period. Such a case can also be solved by an off-line algorithm.

2.3 Results for the Makespan problem with Machine Non-Availability

In the past decades, machine scheduling problems have received much attention by researchers. There are many papers in the literature dealing with parallel machine scheduling. But the number of papers addressing machine scheduling with an availability constraint is very limited. This section first reviews the papers dealing with an availability constraint on a single machine. Then, the result for parallel machines with different machine release times is shown (machines may not be available at time zero). And finally, the last section reviews the literature for parallel machine scheduling with availability constraints within the planning horizon.

When there is a single machine ($1|r-a|C_{max}$), any arbitrary sequence will yield the optimal makespan [25]. However the same result does not hold when other factors are included. Wu and Lee [45] studied this problem with an availability constraint and with deteriorating jobs. Deterioration of a job means that the processing time for a job is a function of its starting time. They solved this problem using binary integer programming technique. Later they showed that the same problem can be solved optimally by the Shortest Processing Time (SPT) rule [46]. The SPT rule is very similar to LPT algorithm, with the exception that jobs are assigned to the machine in the non-decreasing order of their processing time.

Lee [24] studied parallel machine scheduling with availability constraint, where

one of the machines is always available and each machine has at most one availability constraint ($P_m|r - a_{m-1,1}|C_{max}$). He assumed that the durations of the unavailability periods may be different, but they all start at time zero. This problem can also be defined as a classical parallel machines scheduling with machines release times. Based on this assumption, he introduced two algorithms to solve this problem.

In the first algorithm, he assumed that unavailability periods are jobs that are already scheduled on the machines. Then he applies the classical LPT algorithm to assign jobs on the machines; e.g. the first job will be assigned to the machine which is released first. He proves that the makespan of this algorithm is always less than or equal to $\left(\frac{3}{2} - 1/(2m)\right) C^*$. Later, Lee [25] showed by an example that the relative error of this algorithm can be arbitrarily large even for the two-machine problem. He showed that this algorithm performs poorly whenever the start of unavailability period is greater than zero or when there is an availability constraint on all machines.

In some cases a machine might have an unavailability period, which force it to be inactive during the whole scheduling horizon because the duration of unavailability is larger than the current makespan. Lee [27] showed under this circumstance that the LPT bound is not valid anymore. He proved that this bound can be tightened if the number of active umachines (m') is smaller than the total number of machines (m). The performance of the algorithm is then $C_{LPT} \leq \left(\frac{3}{2} - 1/\left(2m'\right)\right) C^*$.

The second method that he applied in the mentioned paper is a modified version of his first algorithm, which he called Modified Longest Processing Time (MLPT). In this method, the non-availability periods are treated as special jobs and they are sorted along with the other jobs in the non-increasing order of their duration. The jobs and unavailabilities are scheduled by the LPT algorithm with the condition that each machine can have at most one special job. So, whenever a machine is assigned more than one special job, the extra special jobs is redirected to the machine with the smallest scheduled processing time and with no special job; then the smallest processing time of that machine is removed and is replaced by the special job. The removed job is then moved to the first of the list of unassigned jobs. He proves that the makespan of this algorithm is bounded by $C_{LPT} \leq \frac{4}{3} C^*$.

Later, Lin et al. [33] improved Lee's bound of $C_{LPT} \leq \frac{4}{3}C^*$ reached by MLPT to $C_{LPT} \leq \frac{5}{4}C^*$. Kellerer [20] developed a dual approximation algorithm using a bin packing approach leading to the same bound.

The papers mentioned above investigate parallel machines scheduling with machine release time, where the unavailability periods may start only at the beginning of the scheduling horizon. This problem is easier than the problem with availability constraint not necessary only at the beginning of the scheduling horizon. This is true because of possibility of having job preemption, when there is availability constraint in the middle of scheduling horizon, which makes the problem more difficult. Lee [25] introduced a new method for $P_m|r - a_{m-1,1}|C_{max}$, named LPT2. This algorithm is very similar to the LPT algorithm. In LPT2, jobs are sorted as in the LPT algorithm but they are assigned to a machine such that the completion time of the job is minimized. Under the assumption that one of the machines is always available, the algorithm yields a performance guarantee of $C_{LPT2} \leq C^* (1 - 1/m) /2 + C^*$.

Liao et al. [31] found the optimal makespan for $P_2|r - a_{1,1}|C_{max}$. They solved this problem by partitioning it into four sub-problems, each of which is solved optimally by an algorithm. But this algorithm works on a very limited model. In their algorithm, there are only 2 machines, and the availability constraint is only on one of the machines. Their algorithm can solve problems up to 100 jobs.

Liao and Sheen [32] considered a problem where the machines may have different job capabilities $P_m|r - M_j - a_{m,1}|C_{max}$. They solved this problem by using a binary search algorithm. The M_j in the beta field denotes the specific subset of machines that can process job j. The algorithm either verifies the infeasibility of the problem or determines the optimal schedule within a predefined planning horizon. Their model is very large and the paper does not report any computational result.

Błażwicz et al. [4] investigated machine scheduling with limited availability and two objective functions of makespan and maximum lateness. They showed that, if the tasks are from chains and are processed by identical processors, then the problem can be solved by low order polynomial time for C_{max} criterion, and a linear programming approach is required for L_{max} criterion. A chain "is a special precedence structure in

which each job has at most one direct predecessor and one direct successor" [3].

When it comes to online algorithm and non-resumable jobs, Tan and He [44] found the optimal solution for $P_2|nr - a_{m,1}|C_{max}$. They scheduled jobs on two identical machines where the unavailability periods of two machines are not overlapping. They showed that the competitive ratio of LS is 3, and the optimal algorithm has a competitive ratio of 5/2. Sleator et al. [43] defined the competitive analysis as a worst case analysis where the performance of an online algorithm is compared to the performance of the optimal offline algorithm [44].

As explained earlier, the solution of makespan problem for one machine is trivial. However this is not true when the jobs are non-resumable. It is shown that, when the jobs are non-resumable, the problem is NP-hard, even for one machine problem with a single availability constraint [25]. When there is more than one availability constraint, the problem is NP-hard in the strong sense. Lee [25] proposed an algorithm for solving the one machine problem with a single availability constraint $(1|nr - a|C_{max})$. He proves that this algorithm has a performance guarantee of $C_{LPT} \leq 4C^*/3$, when the jobs are sorted in the LPT order, and they assigned as many jobs as possible before the start of unavailability period and the rest after the unavailability period.

When there are more than one machine, the problem $P_m|nr - a_{m,1}|C_{max}$ is NP-hard. Lee [25] analyzed the application of LS and LPT algorithm to this problem. He proves that they have a performance guarantee of $C_{LS} \leq C^*m$ and $C_{LPT} \leq C^*(m+1)/2$ respectively.

When the performance measure is the summation of the jobs completion times $(1||\sum C_i)$, the SPT yields the optimal solution. Adiri et al. [1] show that when there is an availability constraint on the machine and the jobs are non-resumable $(1|nr - a|\sum C_i)$, the problem will change to NP-hard. They prove that the SPT rule has a relative error bound of less than equal 1/4. Later, Lee and Liman [28] improved this error bound to 2/7.

Cassady and Kutanoglu [7] studied single machine scheduling with the objective function of minimizing job tardiness. However in their model they do not just schedule jobs. In their study, they assumed that preventive maintenance was not

deterministic, and machine breakdown could happen anytime during the scheduling horizon. Therefore they integrated preventive maintenance planning and production scheduling. They used total enumeration for solving this problem optimally. Their algorithm can not be applied to the problems to more than eight jobs, because the computational time becomes unbearable. Later they studied the same model but with different objective function [8]. They assumed that the objective function is to minimize weighted completion time, and they used the same algorithm to solve the problem. In their paper, a heuristic is also introduced for scheduling large number of jobs. Their heuristic can solve problems up to 20 jobs.

A summary of papers discussed in this chapter can be found in the table 2.1 on the next page.

For more information on machine scheduling with deterministic availability constraints, the reader is referred to the paper by Ma et al. [36].

In the next chapter, we explain the problem and assumptions of the problem. Two Integer Linear Programming (ILP) models are provided to optimally solve the problem with single and multiple availability constraints.

Table 2.1: Summary of Exact and Approximation Algorithms

Problem	Algorithm	Algorithm Performance	References
$1 \mid r - a_i \mid C_{max}$	LPT	Exact	Lee [25]
$1 \mid r - a_i \mid C_{max}$ with deteriorating jobs	Binary Integer Programming	Exact	Wu and Lee [45]
$1 \mid r - a_i \mid C_{max}$ with deteriorating jobs	SPT	Exact	Wu and Lee [46]
$P_m \mid r - a_i \mid C_{max}$, at least one machine is always available and unavilability periods are all start at time zero	LPT	$C_{LPT} \leq \left(\frac{3}{2} - 1/(2m)\right) C^*$	Lee [24]
$P_m \mid r - a_i \mid C_{max}$, at least one machine is always available and unavilability periods are all start at time zero and the number of active machine is smaller than total number of machines	LPT	$C_{LPT} \leq \left(\frac{3}{2} - 1/\left(2m'\right)\right) C^*$	Lee [27]
$P_m \mid r - a_i \mid c_{max}$, at least one machine is always available and unavilability periods are all start at time zero	MLPT	$C_{LPT} \leq \frac{4}{3} C^*$	Lee [24]
$P_m \mid r - a_i \mid c_{max}$, at least one machine is always available and unavilability periods are all start at time zero	MLPT	$C_{LPT} \leq \frac{4}{3} C^*$	Lin et al. [33]
$P_m \mid r - a_i \mid c_{max}$, at least one machine is always available and unavilability periods are all start at time zero	Dual Approximation	$C_{LPT} \leq \frac{4}{3} C^*$	Kellerer [20]

Table 2.1: continued

Problem	Algorithm	Algorithm Performance	References
$P_m \mid r - a_i \mid C_{max}$, at least one of the machine is always available	LPT2	$C_{LPT2} \leq C^* (1 - 1/m)/2 + C^*$	Lee [25]
$P_2 \mid r - a_i \mid C_{max}$, at least one of the machine must always available	Partitioning the problem into four sub-problems solved by TMO-based Algorithm	Exact	Liao et al. [31]
$P_m \mid r - M_j - a_i \mid C_{max}$	Binary Search Algorithm	Exact	Liao and Sheen [32]
$P_m \mid r - a_i j \mid C_{max}$, the tasks are from chains	Muntz and Coffman Algorithm [39]	Exact	Błażwicz et al. [4]
$P_m \mid r - a_i j \mid C_{max}$, the tasks are from chains	Linear Programming Approach	Exact	Błażwicz et al. [4]
$1 \mid nr - a_1 \mid C_{max}$	LPT	$C_{LPT} \leq C^*/3 + C^*$	Lee [25]
$P_2 \mid nr - a_i \mid C_{max}$, at least one of the machine must always available	Partitioning the problem into four sub-problems, solved by TMO-based Algorithm	Exact	Liao et al. [31]
$Pm \mid nr - a_{i1} \mid C_{max}$	LS	$C_{LS} \leq C^* m$	Lee [25]
$Pm \mid nr - a_{i1} \mid C_{max}$	LPT	$C_{LPT} \leq C^*(m+1)/2$	Lee [25]

Chapter 3

Definition and Modelling of the Problem

This chapter describes the problem and the assumptions made in its mathematical modeling. After describing the problem and assumptions, the problem is formulated into mathematical formulae. Two integer linear programming models are proposed to model the makespan minimization for parallel machines with unavailability constraints. The first model assumes that each machine can have at most one availability constraint, while the second model is a generalization of the first model, and each machine can have multiple non-availability periods.

3.1 Model Description

We consider the problem of scheduling N independent jobs (tasks) on m identical machines (processors) with availability constraints. The availability constraint means that some or all machines are not available for a certain period of time and therefore cannot be used to process jobs. The objective is to minimize the maximum completion time (makespan).

In the modeling of this problem, the following assumptions are used:

1) All machines are identical and are able to perform all operations (all eligible).

2) Each machine can process only one task at any time.

3) Each part has only one, maybe complex, operation.

4) Preemption of a job on another machine is not allowed. Operation preemption is allowed (e.g. machines are not all in the same location).

5) All jobs are available at time zero, however some machines may not be available at that time.

6) Setup times are independent of job sequence and are included in the processing times.

7) The unavailability periods are known in advance and their duration are also known and constant. Therefore an off-line algorithm is used to solve the problem.

After the unavailability period the interrupted job can be completed immediately on the same machine. The duration of the interrupted job after the non-availability period depends on the type of the job, e.g. resumable or non-resumable. In this work, we assumed that jobs are resumable.

Since the classical machine scheduling problem $P_m||C_{max}$ is NP-hard [29], it is clear that the generalization problem with machine availability constraints will be also NP-hard.

3.2 Notations

The following notations will be used in the models and the exact algorithm.

m	the number of machines
n	the number of different processing times
R_i	the number of unavailability period(s) on machine i
q, k	the index of unavailability periods
i, α	the index of machines
j, τ, μ	indices of jobs (tasks) and/or processing times
p_j	the j^{th} processing time when all n processing times are sorted in non-increasing order
u_j	the total number of jobs having processing time p_j
N	the total number of jobs $N = \sum_{j=1}^{n} u_j$

v_j	the number of jobs which have processing time p_j and are not loaded on any machine
z_j	the number of jobs with processing time p_j in the load of the current machine.
L_i	the load for machine i.
C_i	the completion time of the last job on machine i
s_{iq}	the starting time of the q^{th} unavailability period on machine i. s_{iq} is equal s_i, when R_i is equal to 1.
e_{iq}	the ending time of the q^{th} unavailability period on machine i. e_{iq} is equal e_i, when R_i is equal to 1.
D_{iq}	the duration of the q^{th} unavailability period on machine i ($D_{iq} = e_{iq} - s_{iq}$). D_{iq} is equal D_i, when R_i is equal to 1.
UB	upper bound of the loads of machines.
LB	lower bound of the load of the machines.
UB_i	upper bound of the load of machine i.
M	an appropriately large positive number.

Under our method, jobs are sorted in the non-increasing order of their processing time. So if we have 7 jobs ($N = 7$) with processing times 2, 5, 2, 6, 5, 8, 5 then we can write $p_1 = 8$, $p_2 = 6$, $p_3 = 5$ and $p_4 = 2$. There are 4 job types ($n = 4$) and $u_1 = 1$, $u_2 = 1$, $u_3 = 3$ and $u_4 = 2$. At any given time, the load of a machine is the set of jobs scheduled or loaded on it. Referring to the list of jobs presented above, if the last three jobs are loaded on machine 1, then the load on machine 1 will be (0(8) 0(6) 1(5) 2(2)). There is 0 job of type 1 (or with processing time 8), 0 job of type 2, 1 job of type 3 (with processing time 5) and 2 jobs of type 4. So $z_1 = 0$, $z_2 = 0$, $z_3 = 1$ and $z_4 = 2$.

The total load of machine 1 is then $0 \times p_1 + 0 \times p_2 + 1 \times p_3 + 2 \times p_4 = 0 \times 8 + 0 \times 0 + 1 \times 5 + 2 \times 2 = 9$.

The load of a machine is then always given by: $\sum_{j=1}^{n} p_j z_j$ and denoted by $(z_1, z_2, \ldots, z_n)$.

3.3 A New Integer Linear Programming Model

A new Integer Linear Programming (ILP) model is developed to optimally schedule N jobs on m machines with up to one non-availability period per machine while minimizing the makespan.

The parameters for the proposed ILP are shown below.

N	number of jobs	$j = 1, \ldots, N$
m	number of machines	$i = 1, \ldots, m$
p_j	processing time for job j	$j = 1, \ldots, N$
s_i	start of unavailability period on machine i	$i = 1, \ldots, m$
e_i	end of unavailability period on machine i	$i = 1, \ldots, m$

In this mathematical model three sets of variables are used:

$$x_{ij} = \begin{cases} 1 & \text{if job } \ j \ \text{ is assigned to machine } i \\ 0 & \text{otherwise} \end{cases}$$

$$y_i = \begin{cases} 1 & \text{if all jobs on machine } i \text{ are completed before } s_i \\ 0 & \text{otherwise} \end{cases}$$

$h =$ the makespan.

This ILP is as follow:

Model 1:

$$min \qquad h \tag{3.1}$$

s.t:

$$\sum_{j=1}^{N} p_j x_{ij} \ \leq \ s_i y_i + M(1 - y_i) \qquad i = 1 \ldots m \tag{3.2}$$

$$\sum_{j=1}^{N} p_j x_{ij} + (e_i - s_i)(1 - y_i) \ \leq \ h \qquad i = 1 \ldots m \tag{3.3}$$

$$\sum_{i=1}^{m} x_{ij} \ = \ 1 \qquad j = 1 \ldots N \tag{3.4}$$

$$x_{ij} \ \in \ \{\, 0, \, 1 \,\} \qquad i = 1 \ldots m, \qquad j = 1 \ldots N \tag{3.5}$$

$$y_i \ \in \ \{\, 0, \, 1 \,\} \qquad i = 1 \ldots m \tag{3.6}$$

$$h \geq 0 \tag{3.7}$$

The unavailability should not be included in the makespan of machine i, when the jobs are completed before the start of the unavailability on that machine. This is guaranteed by constraint (3.2).

Constraint (3.3) assures that the objective function variable is at least as large as the makespan, which should be minimized.

Each task must be assigned to exactly one machine. This is shown by constraint (3.4).

Finally, constraints (3.5)-(3.7) describe the types of the decision variables.

This model has $m(N+1)+1$ decision variables, and $2m+N$ functional constraints.

Model 1 developed above applies to $P_m|r-a_{m,1}|C_{max}$ problems where each machine can have at most one non-availability constraint. Model 1 is a significant contribution to the current state of literature and is optimally solvable by ILP solvers for moderately sized problems as will be shown in Chapter 5. In industrial settings, it is not uncommon to have several non-availability periods per machine over the scheduling horizon. In order to address this more general problem, model 1 is to be extended to include multiple non-availability periods per machine.

3.4 An ILP for Multiple non-availability Periods (Model 2)

It is common to encounter several non-availability periods on a machine as a result of recurring periodic preventive maintenance actions, several special jobs being booked and/or a combination of both. Adding more non-availability periods adds to the complexity and difficulty of the model. In this section a new ILP is proposed to model the makespan minimization problem on parallel machines with multiple availability constraints per machine. Using Graham's taxonomy this problem is denoted by $P_m|r-a_{m,q}|C_{max}$. First, a small problem with one machine and three non-availability periods is used to illustrate the mathematical model. Then, the generalized model is provided.

Assume we want to solve by the ILP the following problem:

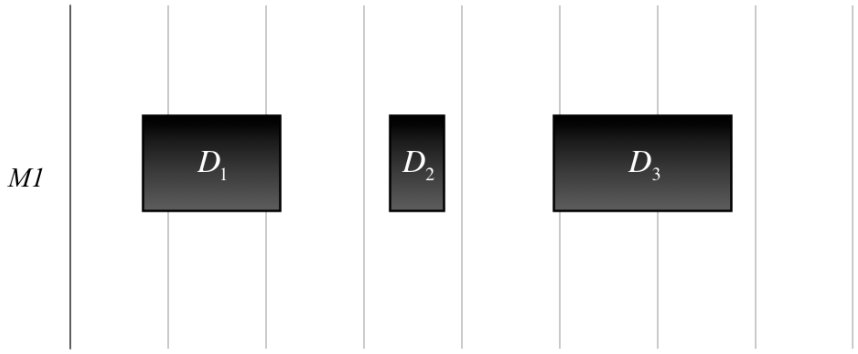

Figure 3.1: A Problem with Multiple Availability Constrains

As can be seen from the above figure, the problem is the problem of scheduling jobs on a machine with three unavailability periods denoted by D_1, D_2, D_3. Let R_i denote the number of non-availability periods on machine i. A binary variable y_{iq} is introduced in the model.

$$y_{iq} = \begin{cases} 1 & \text{if jobs on machine } i \text{ are completed before } s_{iq} \\ 0 & \text{otherwise} \end{cases}$$

The ILP for this problem is as follows:

$$min \qquad h \tag{3.8}$$

s.t:

$$\sum_{j=1}^{N} p_j x_{1j} + (e_{11} - s_{11})\, y_{11} + (e_{11} - s_{11} + e_{12} - s_{12})\, y_{12} \;\leq \\ s_{11}y_{11} + s_{12}y_{12} + s_{13}y_{13} + M\,(1 - (y_{11} + y_{12} + y_{13})) \tag{3.9}$$

$$\sum_{j=1}^{N} p_j x_{1j} + (e_{11} - s_{11})\, y_{12} + (e_{11} - s_{11} + e_{12} - s_{12})\, y_{13} + \\ (e_{11} - s_{11} + e_{12} - s_{12} + e_{13} - s_{13})\,(1 - (y_{11} + y_{12} + y_{13})) \;\leq\; h \tag{3.10}$$

$$x_{1j} \;=\; 1 \quad j = 1 \ldots N \tag{3.11}$$

$$y_{11} + y_{12} + y_{13} \;\leq\; 1 \quad j = 1 \ldots N \tag{3.12}$$

$$x_{1j} \;\in\; \{\, 0,\, 1 \,\} \quad j = 1 \ldots N \tag{3.13}$$

$$y_{1R_i} \;\in\; \{\, 0,\, 1 \,\} \quad i = 1 \ldots m,\, \forall R_i \tag{3.14}$$

$$h \geq 0 \tag{3.15}$$

Let us define $D_{ik} = e_{ik} - s_{ik}$ for all i and k, then the general model is given by

Model 2:

$$min \qquad h \tag{3.16}$$

s.t:

$$\sum_{j=1}^{N} p_j x_{ij} + \sum_{q=1}^{R_i-1} \left(\sum_{k=1}^{q} D_{ik} \right) y_{iq} \;\leq\; \sum_{q=1}^{R_i} s_{iq} y_{iq} + M \left(1 - \left(\sum_{q=1}^{R_i} y_{iq} \right) \right) \qquad i = 1 \ldots m \tag{3.17}$$

$$\sum_{j=1}^{N} p_j x_{ij} + \sum_{q=1}^{R_i-1} \left(\sum_{k=1}^{q} D_{ik} \right) y_{iq+1} + \left(\sum_{q=1}^{R_i} D_{iq} \right) \left(1 - \left(\sum_{q=1}^{R_i} y_{iq} \right) \right) \;\leq\; h \qquad i = 1 \ldots m \tag{3.18}$$

$$\sum_{i=1}^{m} x_{ij} \;=\; 1 \quad j = 1 \ldots N \tag{3.19}$$

$$\sum_{q=1}^{R_i} y_{iq} \;\leq\; 1 \quad i = 1 \ldots m \tag{3.20}$$

$$x_{ij} \;\in\; \{\, 0,\, 1 \,\} \quad i = 1 \ldots m,\, j = 1 \ldots N \tag{3.21}$$

$$y_{iq} \;\in\; \{\, 0,\, 1 \,\} \quad i = 1 \ldots m,\, q \in R_i \tag{3.22}$$

$$h \geq 0 \tag{3.23}$$

Constraint (3.20) guarantees that no more than one y_{iq} is equal one. The rest of the constraints can be interpreted as in model 1.

The above ILP has $\sum_{i=1}^{m} R_i + mN + 1$ decision variables, and $(3m + N)$ functional constraints.

Note that model 2 can be converted to model 1, when the number of unavailabilities on each machine is at most one ($R_i \leq 1 : \forall i$).

The next chapter explains the proposed algorithm, which is developed to find optimal solution for parallel machine scheduling with availability constraints. The performance of the algorithm is evaluated by extensive numerical studies, and the results show the efficiency of the algorithm for solving these problems with superior Central Processing Unit time (CPUt).

Chapter 4

Model Development

The ILP models introduced in the previous chapter can be optimally solved by the ILP solvers for small size problems. For model 1, CPLEX can solve problems with up to 50 machines and 101 jobs. An exact algorithm has been developed for large and more complicated problems. The proposed algorithm, solves the scheduling of parallel machines with multiple availability constrains $P_m|r - a_{i,q}|C_{max}$ optimally.

In this chapter, the first section introduce the notations along with the basic tools needed to develop the main algorithm. The second section describes the method employed to solve the problem. Section three describes how an initial upper bound is developed for the algorithm. The proposed exact algorithm is introduced in section 4.4. With some modifications, we can apply the algorithm to the problem with multiple availability constraints. This is explained in section 4.5. Lemmas are defined to limit the search space and decrease the computational requirements of the algorithm. These lemmas are defined in section 4.6. Finally to give a better understating of the algorithm to the reader, a numerical example is constructed and shown in the section 4.7.

4.1 Preliminary Presentation of the Tools Used in the Algorithm

This section goes over the preliminary tools require for building the main algorithm.

The objective function value of any feasible solution, i.e. the current C_{max}, is the load of the machine that finishes last. Thus

$$C_{max} = max\left\{C_i : i = 1 \dots m\right\} \tag{4.1}$$

where C_i is the completion time of the last job on machine i

Assume that the makespan of the current known feasible solution is C^{best}. If the optimal solution has not been explored yet then its value is not greater than $\left(C^{best} - 1\right)$ in the case of integer processing times. Thus

$$UB = C^{best} - 1 \tag{4.2}$$

is an upper bound for the load of all machines in the feasible solutions still to be investigated. Hence

$$LB \;=\; \max \left\{ 0, \sum_{j=1}^{N} p_j - (m-1)UB \right\} \tag{4.3}$$

is a lower bound for all the other loads in the same feasible solution. Equation (4.3) implies that if all machines but one have a total load equal to the upper bound, then the remaining load is the lower bound. In this formula, N is the number of jobs, p_j is the processing time of job j and m is the number of machines. After finding a new and better feasible solution both bounds tighten up.

Equation (4.2) is a global upper bound on the current makespan of all machines. However, the unavailability periods on a machine can affect this upper bound. For example, we may not be able to load the same number of jobs on two machines, when they have different availability constraints. Therefore another upper bound is defined to measure the maximum load on each machine that takes into account the duration of the non-availability period. Thus

$$UB_i \;=\; \begin{cases} UB & UB \leq s_i \\ UB - min\{UB, e_i\} + s_i & UB \geq s_i \end{cases} \tag{4.4}$$

The global upper bound and the machine upper bound are equal when the unavailability period is before the upper bound. When the unavailability period is within or before the upper bound, the upper bound for the machine is equal to the global upper bound minus the duration of the non-availability period.

It is possible that an availability constraint on a machine does not impact or affect the load on the machine. For example, when the jobs are completed before the start

of the first unavailability period on the machine. Thus, the effective duration of the unavailability period on machine i can be calculated by:

$$D_i = \begin{cases} UB - s_i & \text{if} \quad s_i \leq UB \leq e_i \\ e_i - s_i & \text{if} \quad UB > e_i \end{cases} \tag{4.5}$$

That is the duration of the unavailability on each machine is only equal to the period of time during which the machine is not available within the global upper bound.

4.2 Lexicographic Order of the Machine Loads

In order to find an optimal solution, the exact algorithm should have two important properties. First, the method should not be computationally complex. This will enable the algorithm to solve large-scale or industrial problems. Second, the method should enumerate in an implicit or explicit way all possible loads, when assigning jobs to the machines. This ensures that the algorithm evaluates all feasible solutions.

As explained before, the main algorithm is an implicit enumeration. It is very important to ensure that the enumeration does not skip any potential feasible solution. The load of the machines are determined in sequence, one after another. Thus, the potential load of machine i with $1 < i \leq m$ depends on the loads of the previous machines.

Any load on a machine must satisfy two constraints:

(i) it cannot contain more jobs than the number of remaining jobs,

(ii) the total load, i.e. the sum of the processing times, must be between the lower and upper bounds.

These two conditions can be described by the following system of inequalities:

$$0 \leq z_j \leq v_j \quad j = 1, \ldots, n \tag{4.6}$$

$$LB \leq \sum_{j=1}^{n} p_j z_j \leq UB \tag{4.7}$$

$$z_j \text{ is integer } \; j = 1, \ldots, n. \tag{4.8}$$

where z_j is the number of jobs with processing time p_j in the load of the current machine, n is the number of different processing times, v_j is the number of tasks with processing time p_j that are still not loaded on any machine. For example in the case of the first machine $v_j = u_j$ for all j (u_j is the number of unscheduled jobs), but the same statement is not true for the other machines as some v_j are strictly less than the corresponding u_j according to the load of the first machine.

There are many feasible solutions to the system (4.6)-(4.8). For an implicit enumeration procedure, they must be ordered somehow and enumerated in this order. One easy way to perform this task is to order them lexicographically. Lexicographical order, also known as the dictionary order or the alphabetic order, can be defined as the cartesian product of any two ordered sets of X and Y. In other words:

$$\begin{aligned} &\text{if } (x_0, y_0), (x, y) \in X \times Y \text{ then } (x_0, y_0) < (x, y) \text{ iff either} \\ &\qquad \text{(i) } x_0 < x, \textit{or} \\ &\qquad \text{(ii) } x_0 = x \text{ and } y_0 < y \end{aligned} \tag{4.9}$$

For example the lexicographical order of $1, 1, 2$ is:

#1:	1	1	2	#7:	0	1	2
#2:	1	1	1	#8:	0	1	1
#3:	1	1	0	#9:	0	1	0
#4:	1	0	2	#10:	0	0	2
#5:	1	0	1	#11:	0	0	1
#6:	1	0	0	#12:	0	0	0

Lexicographical search has been widely used in the literature in different ways. Ho and Wong [19] used a lexicographic search, to optimally solve the scheduling of jobs on two machines ($P_2||C_{max}$). Later, their method was used by Liao et al. [31] to optimally schedule jobs on two machines, where one of the machines has an availability constraint ($P_2|a - r_{1,1}|C_{max}$). Hashemian and Vizvári [18] scheduled both

production and vehicles on a special flexible manufacturing system, where jobs are scheduled according to the lexicographical order.

The largest solution in the lexicographical order for n job types is given by (4.10)-(4.11). The load is determined by a greedy algorithm:

$$z_1 \;=\; \min\left\{ \left\lfloor \frac{UB}{p_1} \right\rfloor,\, v_1 \right\} \tag{4.10}$$

$$z_j \;=\; \min\left\{ \left\lfloor \frac{UB - \sum_{\beta=1}^{j-1} p_\beta z_\beta}{p_j} \right\rfloor,\, v_j \right\} \quad j = 2, \ldots, n. \tag{4.11}$$

In the lexicographic order, the next load is needed in the enumeration whenever an infeasible branch is explored. It can be determined by the greedy method in the following way, where the current load is z_j, and the lexicographic next is $\bar{z}_j$

$$\gamma \;=\; \max\{\, j \mid z_j \;>\; 0 \,\} \tag{4.12}$$

$$\bar{z}_j \;=\; z_j \quad j \;=\; 1, \ldots, \gamma - 1 \tag{4.13}$$

$$\bar{z}_\gamma \;=\; z_\gamma \;-\; 1 \tag{4.14}$$

$$\bar{z}_j \;=\; \min\left\{ \left\lfloor \frac{UB - \sum_{\beta=1}^{j-1} p_\beta z_\beta}{p_j} \right\rfloor,\, v_j \right\} \quad j = \gamma + 1, \ldots, n. \tag{4.15}$$

There is no further feasible load of the machine if

$$\forall\, j:\; z_j > 0 \text{ implies that } \sum_{\beta=1}^{j-1} p_\beta z_\beta \;+\; p_j(z_j - 1) \;+\; \sum_{\beta=j+1}^{n} p_\beta v_\beta \;<\; LB. \tag{4.16}$$

The greedy algorithm goes through multiple iterations. An interval of minimum and maximum load of a machine is constructed for the initial iteration, and whenever a feasible schedule of machines is obtained in an iteration, the algorithm will move to the next iteration. After moving to the new iteration, the global upper bound and lower bound are updated immediately. At each iteration, the lower bound and upper bound will tighten up, until the algorithm eventually finds the optimal solution for the makespan.

4.3 Initial Solution

According to the equation (4.2), the global upper bound at each iteration is equal to the makespan of the previous iteration (solution) minus one. However, there is no feasible solution at the very first iteration that can be used to create the upper bound. For this purpose, an upper bound for the first iteration is obtained by an existing heuristic. Among the heuristics available in the literature, LPT2 [25] results in relatively good performance guarantee compared to other heuristics. Therefore, LPT2 is employed for constructing the initial solution. This algorithm always assigns the next job to the machine which can minimize its completion time. Assume that L_i is the load on machine i, then the next job j will be scheduled on machine i^* such that:

$$i^* = argmin\{L_i + p_j : i = 1, 2, \ldots, m\} \tag{4.17}$$

Summary of LPT2 algorithm is given below:

Algorithm 1:

step 1. Sort the N jobs according to the non-increasing order of their processing time.

step 2. Set $j = 1$.

step 3. Assign job j to machine i according to equation (4.17).

step 4. If $j = N$ (i.e. all jobs are allocated) then go to the next step, otherwise set $j = j + 1$ and go to step 3.

step 5. Calculate C_{max}^{LPT2} by using equation (4.1).

4.4 The Main Algorithm

The main algorithm is an enumeration, which consists of two types of operations: construction, and backtrack.

4.4.1 Construction

In the construction phase the algorithm determines the load of the machines one by one. This is done by either equations (4.10)-(4.11) if the machine has no previous load or by the formulae (4.12)-(4.15) if it had a load. Whenever a potential load is determined for machine i, the feasibility of the load is checked by verifying that the total processing time is at least as high as the lower bound and less or equal to the upper bound. If the feasibility condition is violated, then again the next possible load is determined by formulae (4.12)-(4.15). If the construction is successful i.e. all machines are loaded and all conditions are satisfied, then the lower bound, global upper bound and upper bounds of the machines are updated. The enumeration is continued on the machine, which has maximal load and smallest index.

Its lexicographically largest load is determined for the new upper bound according to formulae (4.10)-(4.11).

4.4.2 Backtracking

The algorithm is always looking for the optimal solution. Therefore, it is really important not to miss any possible solution. For this purpose, backtracking is applied in the scheduling algorithm to allow the updating of the scheduling decisions.

In this algorithm, the backtracking is applied whenever any of the following two situations are accounted:

1. When the load of a machine is not feasible, e.g. the load cannot satisfy equation (4.7). At this point, based on the stage of the scheduling, the backtracking will be done either on machine i or machine $i-1$.

2. When a new feasible solution has been found for all machines, e.g. updating makespan. At this point the backtracking will be done on the machine with maximum load. Thus

$$i^{**} = argmax\left\{C^i : i = 1, 2, \ldots, m\right\} \tag{4.18}$$

where

i^{**} is the number or index of machine to which the backtracking is applied

C^i is the completion time of the last job on machine i.

A Tie is broken by selecting the machine with the smallest index.

If the machine has no further feasible load then a backtrack must be made at machine $i-1$, where i is not equal to 1. If i is equal to 1, then there is no feasible solution for the current upper bound and the optimal makespan is equal to the current global upper bound plus one.

4.4.3 The Exact Algorithm

The exact algorithm is described below, where l is the index of the iterations.

step 1. Set $i = 1$ and $l = 1$.

step 2. Use Algorithm 1 (LPT2) to find the initial upper bound.

step 3. Calculate the upper bounds and duration of unavailability periods for unloaded machines by equation (4.4) and (4.5) respectively.

step 4. Load machine i according to the formulae (4.10)-(4.11).

step 5. If machine i does not satisfy inequality (4.7) go to *step 9*.

step 6. If $i \neq m$ then go to *step 8* (not all machines are loaded).

step 7. Set $UB = C_{max} - 1$ and $l = l + 1$. Find i^{**} by using equation (4.18). If $i^{**} = m$ then set $i = i^{**} - 1$; otherwise set $i^{**} = i$. Go to *step 3*.

step 8. Set $i = i + 1$. If $i = m$ then allocate the remaining load on machine i and go to *step 5*; otherwise go to *step 4*.

step 9. If $\sum_{j=1}^{n} z_j = 0$ then set $m = m - 1$ and go to *step 11*;

step 10. Load machine i according to the formulae (4.12)-(4.15), and go to *step 5*.

step 11. If $m > 0$ then go to *step 10*.

step 12. Set $C^*_{max} = UB + 1$.

It should be noticed that at the end of the algorithm, the number of generated feasible solution(s) is $l - 1$. Each one of them is better than the previous one. If $l = 1$ then, the initial solution is obtained by algorithm 1 (LPT2) is optimal.

The figure on the next page shows the flowchart of this algorithm.

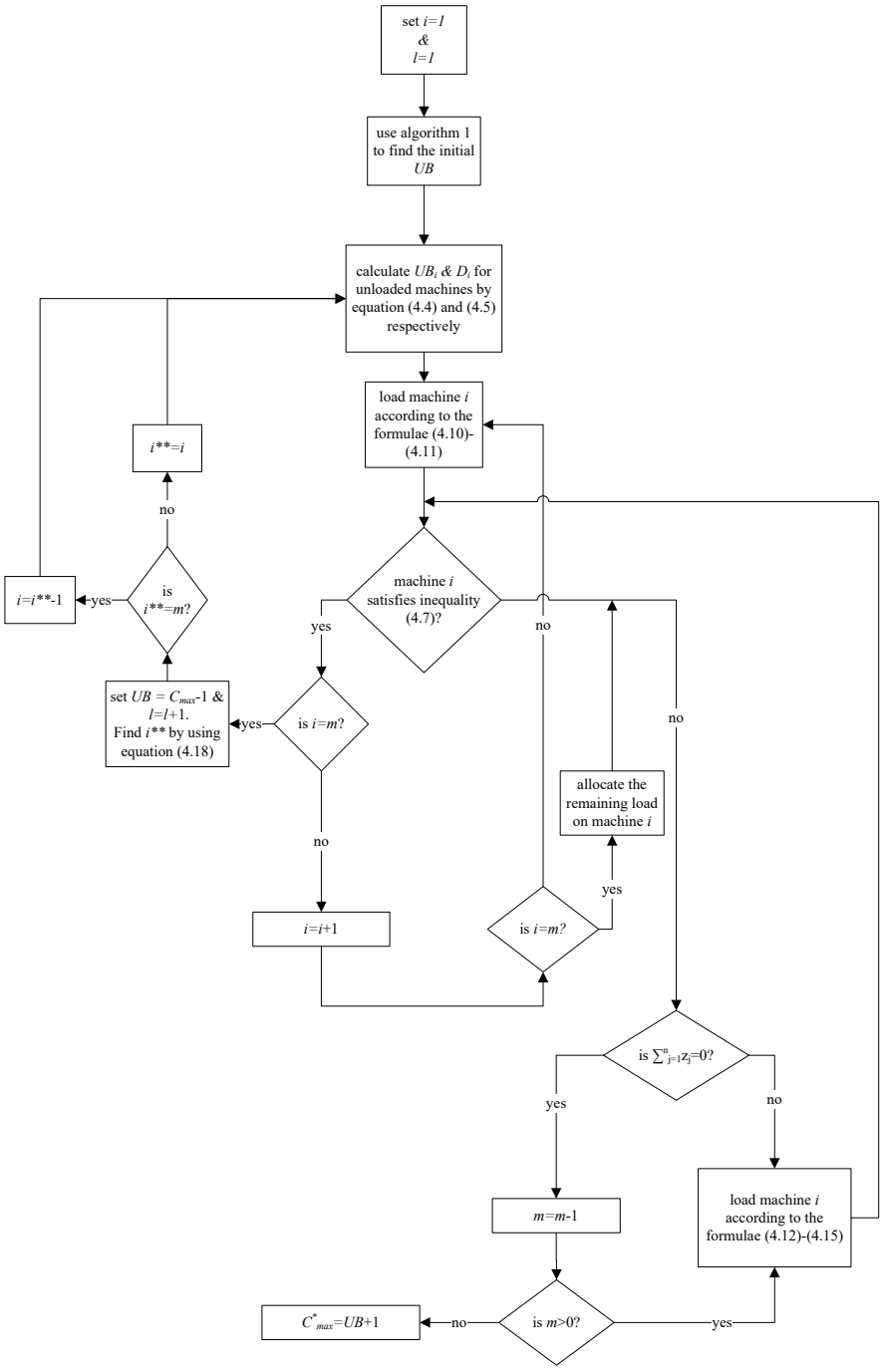

Figure 4.1: The Flowchart for the Exact Algorithm

4.5 Multiple Availability Constraint

The algorithm is built in a way that with some modifications it can also be applied to the problem with multiple availability constraints on each machine. These modifications are shown below.

Since the number of availability constraints on a machine can be greater than one, equation (4.4) should be changed to

$$q^* = max\left\{q : s_{i,q} < UB\right\} \tag{4.19}$$

$$UB_i \;=\; \begin{cases} UB & q_i^* = \emptyset \\ UB - min\{UB, e_{iq^*}\} - \sum_{q=1}^{q^*-1} e_{iq} + \sum_{q=1}^{q^*} s_{iq} & otherwise \end{cases} \tag{4.20}$$

As in the first model, the total duration of unavailabilities on each machine can be calculated by:

$$q^* = max\left\{q : UB > e_{i,q}\right\} \tag{4.21}$$

$$D_i \;=\; \begin{cases} UB - s_{i1} & q^* = \emptyset \\ UB - min\left\{UB, s_{i,q^*+1}\right\} + \sum_{q=1}^{q^*}\left(e_{iq} - s_{iq}\right) & q^* < R_i \\ \sum_{q=1}^{q^*}\left(e_{iq} - s_{iq}\right) & q^* = R_i \end{cases} \tag{4.22}$$

The rest of the algorithm remains the same as in the model for the single availability constraint.

4.6 Methods to Accelerate the Algorithm

For the effectiveness of the method it is crucial to find feasible solutions as soon as possible. For this reason the following lemmas are defined to limit the search space. Except for lemma 4.6.2 which reduces the number of infeasible solutions, the rest of the lemmas are used to find an infeasible load for a machine, which might be feasible for now, but will affect the feasibility of the load of another machine. When the load of a machine is strictly less than the current upper bound then there is a loss in the

total capacity. These losses can accumulate and result in an infeasibility as shown by the next lemma.

Lemma 4.6.1. *Assume that the loads of the first α machines $(1 \leq \alpha < m)$ are determined and the loads are $L_i\,(i = 1, 2, \ldots, \alpha)$. If*

$$\frac{\sum_{j=1}^{N} p_j - \sum_{i=1}^{\alpha} L_i + \sum_{i=\alpha+1}^{m} D_i}{m - \alpha} \; > \; UB \tag{4.23}$$

then at least one of the unloaded machines must have an infeasible load.

Proof. The left-handed side of the inequality is the average load of the machines remaining to be loaded. When this quantity is higher than the allowed upper bound, at least one machine must have a load greater than the upper bound. □

If the inequality of the lemma is satisfied then a backtrack step must immediately be executed in the algorithm.

When (4.23) is satisfied, then the schedule is infeasible and this is because the total processing time on the currently loaded machines is not large enough. This observation leads to two important results explained in the following lemmas.

Lemma 4.6.2. *Let $\alpha < m$ be the index of the machine currently loaded. Assume that there are n jobs types, where n is a job type with the smallest processing time. Assume further that the jobs are scheduled in LPT order and the load on machine α satisfies inequality (4.23), then equation (4.12) will be changed to*

$$\gamma \; = \; \max\{\, j \mid z_j \; > \; 0 \quad \textit{and} \quad j \neq n \,\} \tag{4.24}$$

Proof. If we don't change the first $n-1$ job types then the load on machine α is always decreasing. Therefore there is no way to find a feasible load (larger than the current infeasible load) for machine α, without changing any of the first $n-1$ job types. □

Lemma 4.6.3. *If*

$$\frac{\sum_{j=1}^{N} p_j - \sum_{i=1}^{\alpha-1} L_i + UB_\alpha + \sum_{i=\alpha+1}^{m} D_i}{m-\alpha} > UB \tag{4.25}$$

then there is no feasible load for machine α and backtrack step should be applied.

Proof. The left-hand side of the inequality is the average load of the machine(s) which are still not loaded, when the load on machine α is equal to the UB_α. If this quantity is higher than the allowed upper bound, then there is no feasible load for machine α. □

Lemma 4.6.3 does not take into consideration how large the violation of the upper bound is. If the violation is large enough, then there will be no feasible load for the current upper bound. The next lemma shows how large this violation must be to cause infeasibility on the current load.

Lemma 4.6.4. *If*

$$\frac{\sum_{j=1}^{N} p_j - \sum_{i=1}^{\alpha} UB_i + \sum_{i=\alpha+1}^{m} D_i}{m-\alpha} > UB \tag{4.26}$$

then there is no feasible solution for the given UB, and C^*_{max} is equal to $UB+1$.

Proof. The left-hand side of the inequality is the average load of the machines which are still not loaded, when the scheduled machine(s) are loaded to their maximum. If this quantity is higher than the allowed upper bound, then there is no feasible schedule for the current upper bound. □

Lemma 4.6.5. *Let α be the index of the currently loaded machine and b_j be the number of jobs with processing time p_j on the load of machine $\alpha-1$. Assume that its load is not feasible and a backtrack to machine $\alpha-1$ has to be carried out, then if*

$$\mho = max\{j | b_j > 0 \text{ and } p_j(z_j + v_j + 1) \leq UB\} \qquad j = 1, 2, \ldots, n \tag{4.27}$$

then the load on machine α may become feasible iff the new load on machine $\alpha-1$ is greater than its current load, or the result of (4.27) is nonempty.

Proof. Assume that the load on machine α is infeasible and we need to backtrack to machine $\alpha - 1$. According to the logic of the algorithm, the smallest job on machine $\alpha - 1$ should be removed, and should be scheduled on machine α. Therefore we can only check the effect of this new load on machine α, which is done by equation (4.27). □

Lemma 4.6.6. *Assume the result of (4.27) is equal to n and the new load on machine $\alpha - 1$ is not greater than its previous load. Then a feasible load on machine α might be found only when job type n is fixed and is equal to all the unscheduled jobs of this type.*

Proof. All the loads on machine α have been checked, except for the $z_n = v_n$. Therefore, we can only search for a feasible solution in the defined by $z_n = v_n$. □

4.7 A Numerical Example

To show how the algorithm works, an example is illustrated and solved. Ten jobs have to be scheduled on 3 machines each having one non-availability period. The 10 jobs have 4 different processing times: two jobs have processing time 18, two other jobs have processing time 17, two jobs have processing times 16 and and the four last jobs have processing time 10.

The non-availability periods start at instant 0, 6, 15, last 5, 6, and 4 unit of time on machine 1, 2, and 3 respectively.

$$
\begin{array}{llllll}
m = 3 & N = 10 & n = 4 & R_1 = 1 & R_2 = 1 & R_3 = 1 \\
p_1 = 18 & p_2 = 17 & p_3 = 16 & p_4 = 10 & & \\
u_1 = 2 & u_2 = 2 & u_3 = 2 & u_4 = 4 & & \\
s_1 = 0 & s_2 = 6 & s_3 = 15 & & & \\
e_1 = 5 & e_2 = 12 & e_3 = 19 & & &
\end{array}
$$

The first step in solving this problem is the construction of the initial solution by LPT2.

The following Gantt chart shows the schedule obtained with LPT2. The solution obtained by this algorithm is 59 $\left(C_{max}^{LPT2} = 59\right)$.

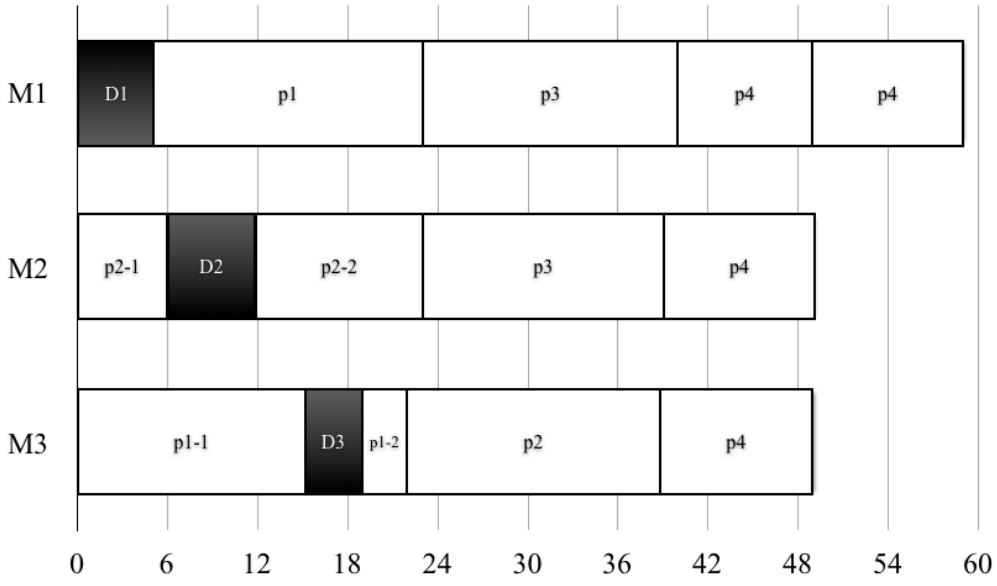

Figure 4.2: Schedule Obtained by the LPT2

The upper bound for each machine can be calculated, with using formula (4.4). Thus

$$UB_1 = 59 - min\{59, 5\} + 0 = 54$$
$$UB_2 = 59 - min\{59, 12\} + 6 = 53$$
$$UB_3 = 59 - min\{59, 19\} + 15 = 55$$

Formula (4.5) is used to calculate the duration of the unavailability on each machine.

$$D_1 = 5 - 0 = 5 \qquad D_2 = 12 - 6 = 6 \qquad D_3 = 19 - 15 = 4$$

Once the upper bound and the duration the unavailability period are known for each machine, the load on the first machine can be determined by formulae (4.10)-(4.11). The load for machine 1 is shown below.

M1: 2(18) 1(17) 0(16) 0(10) $\qquad L_1 = 53 \qquad C_1 = 58$

The feasibility of load on machine 1 can be checked by using formula 4.6.1.

$\frac{142-53+10}{3-1} = 49.5 < 59$ Therefore the load on machine 1 is feasible and we proceed to the next machine.

The load on machine 2 is:

M2: 0(18) 1(17) 2(16) 0(10) $L_2 = 49$ $C_1 = 55$

$\frac{142-102+4}{3-2} = 44 < 59$ Therefore the load on machine 2 is feasible and we proceed to the next machine.

Since machine 3 is the last machine, all remaining jobs should be scheduled on that machine.

M3: 0(18) 0(17) 0(16) 4(10) $L_3 = 40$ $C_3 = 44$

The current $C_{max} = max\{C_1; C_2; C_3\} = 58$. Therefore we update the global upper bound using formula (4.2).

$$UB = 58 - 1 = 57$$

After updating the global upper bound, the upper bound and duration of unavailability for each machine are determined. The same formulae used in the previous iteration are used here again.

$UB_1 = 57 - min\{57, 5\} + 0 = 52$
$UB_2 = 57 - min\{57, 12\} + 6 = 51$
$UB_3 = 57 - min\{57, 19\} + 15 = 53$

$D_1 = 5 - 0 = 5$ $D_2 = 12 - 6 = 6$ $D_3 = 15 - 19 = 4$

Whenever the upper bound is updated, backtracking is applied. From equation (4.18), we can see that the backtracking should be started on the machine with the largest makespan and minimum index. Therefore in this iteration backtracking should be started on machine 1.
The load on machine 1 is determined by formulae (4.10)-(4.11)

M1: 2(18) 0(17) 0(16) 0(10) $L_1 = 36$ $C_1 = 36 + 5 = 41$

$\frac{142 - 41 + 10}{3 - 1} = 55.5 < 57$ The load is feasible and we proceed to the next machine.

M2: 0(18) 2(17) 1(16) 0(10) $L_2 = 50$ $C_2 = 50 + 6 = 56$

$\frac{142 - 86 + 4}{3 - 2} = 60 > 57$ The load is infeasible and we need to proceed to lemma 4.6.4

$\frac{142 - 103 + 4}{3 - 2} = 43 > 57$ The lemma shows that there might still be a feasible load for the current upper bound.

After checking the optimality of the makespan, we need to check the feasibility of other possible loads on machine 2. If lemma 4.6.3 is satisfied, then there is no feasible load for machine 2.

$\frac{142 - (36 + 51) + 4}{3 - 2} = 59 > 57$ Therefore there is no feasible load for machine 2, and backtracking is applied on machine 1.

The load on machine 1 can be modified by using formulae (4.12)-(4.15).

M1: 1(18) 2(17) 0(16) 0(10) $L_1 = 52$ $C_1 = 52 + 5 = 57$

$\frac{142 - 52 + 10}{3 - 1} = 50 < 57$ The load is feasible and we proceed to the next machine.

M2: 1(18) 0(17) 2(16) 0(10) $L_1 = 50$ $C_1 = 50 + 6 = 56$

$\frac{142 - (52 + 50) + 4}{3 - 2} = 44 < 57$ The load is feasible and we proceed to the last machine.

M3: 0(18) 0(17) 0(16) 4(10) $L_3 = 40$ $C_3 = 40 + 4 = 44$

The data are updated as follow:

$$UB = 57 - 1 = 56$$

$UB_1 = 56 - min\{56, 5\} + 0 = 51$
$UB_2 = 56 - min\{56, 12\} + 6 = 50$
$UB_3 = 56 - min\{56, 19\} + 15 = 52$

$D_1 = 5 - 0 = 5$ $D_2 = 12 - 6 = 6$ $D_3 = 15 - 19 = 4$

According to the formula 4.18 the scheduling should be started from the machine 1.

M1: 2(18) 0(17) 0(16) 0(10) $L_1 = 36$ $C_3 = 36 + 5 = 41$

$\frac{142 - (36) + 10}{3 - 1} = 58 > 56$ The load is infeasible and we proceed to the lemma 4.6.4.

$\frac{142 - (51 + 50 + 52) + 10}{3 - 1} = 1.33 < 56$ The lemma shows that there might still be a feasible load for the current upper bound.

$\frac{142 - 51 + 10}{3 - 1} = 50.5 > 56$ Therefore the backtracking step is carried back to machine 1.

M1: 1(18) 1(17) 1(16) 0(10) $L_1 = 51$ $C_1 = 51 + 5 = 56$

$\frac{142 - 51 + 10}{3 - 1} = 50.5 < 56$ The load is feasible and we proceed to the machine 2.

M2: 1(18) 1(17) 0(16) 1(10) $L_2 = 45$ $C_2 = 45 + 6 = 51$

$\frac{142 - (51 + 45) + 4}{3 - 2} = 50 < 56$ The load is feasible and we proceed to the machine 3.

M3: 0(18) 0(17) 1(16) 3(10) $L_3 = 46$ $C_3 = 46 + 4 = 50$

As with the previous iteration the data are updated as follow:

$$UB = 56 - 1 = 55$$

$UB_1 = 55 - min\{55, 5\} + 0 = 50$
$UB_2 = 55 - min\{55, 12\} + 6 = 49$
$UB_3 = 55 - min\{55, 19\} + 15 = 51$

$D_1 = 5 - 0 = 5$ $D_2 = 12 - 6 = 6$ $D_3 = 15 - 19 = 4$

According to formula 4.18 the scheduling should be started from the machine 1.

M1: 2(18) 0(17) 0(16) 1(10) $L_1 = 46$ $C_1 = 46 + 5 = 51$

$\frac{142 - 46 + 10}{3 - 1} = 53 < 55$ The load is feasible and we proceed to machine 2.

M2: 0(18) 2(17) 0(16) 1(10) $L_2 = 44$ $C_2 = 44 + 6 = 50$

$\frac{142 - (46 + 44) + 4}{3 - 2} = 56 > 55$ The load is infeasible and we proceed to lemma 4.6.4.

$\frac{142-(50+49)+4}{3-2} = 47 < 56$ The lemma shows that, there might still be a feasible load for the current upper bound.

$\frac{142-(46+49)+4}{3-2} = 51 < 56$ Therefore there might still be feasible load for machine 2.

Since machine 2, is already loaded and has an infeasible load, we need to reload this machine, with respect to lemma 4.6.2 and formulae 4.12-4.15.

M2: 0(18) 1(17) 2(16) 0(10) $L_2 = 49$ $C_2 = 49 + 6 = 55$

$\frac{142-(46+49)+4}{3-2} = 51 < 55$ The load is feasible and we proceed to the next machine.

M3: 0(18) 1(17) 0(16) 3(10) $L_3 = 47$ $C_3 = 47 + 4 = 51$

With respect to the new upper bound, the data are updated.

$$UB = 55 - 1 = 54$$

$UB_1 = 54 - min\{54, 5\} + 0 = 49$
$UB_2 = 54 - min\{54, 12\} + 6 = 48$
$UB_3 = 54 - min\{54, 19\} + 15 = 50$

$D_1 = 5 - 0 = 5$ $D_2 = 12 - 6 = 6$ $D_3 = 15 - 19 = 4$

In the previous iteration, machine 2 had the maximum load, therefore the backtracking should be started from that machine.

M2: 0(18) 2(17) 0(16) 1(10) $L_2 = 44$ $C_2 = 44 + 6 = 50$

$\frac{142-(46+44)+4}{3-2} = 56 > 54$ The load is infeasible and we proceed to the lemma 4.6.4.

$\frac{142-(49+48)+4}{3-2} = 49 < 54$ Therefore the optimality of the solution has not been proved yet, and we need to proceed to the next lemma.

$\frac{142-(46+48)+4}{3-2} = 52 < 54$ Therefore the other solution for machine 2 should be explored.

M2: 0(18) 1(17) 1(16) 1(10) $L_2 = 43$ $C_2 = 44 + 6 = 50$

Without checking the lemma, we can conclude that the load is not feasible. Since the current load on machine 2 is smaller than the its infeasible previous load. Therefore, we need to reload machine 2.

M2: 0(18) 1(17) 0(16) 3(10) $L_2 = 47$ $C_2 = 47 + 6 = 53$

$\frac{142-(46+47)+4}{3-2} = 53 < 54$ The load is feasible and we schedule the rest of the jobs on the last machine.

M3: 0(18) 1(17) 2(16) 0(10) $L_3 = 49$ $C_4 = 49 + 4 = 53$

With respect to the new upper bound, the data are updated.

$$UB = 53 - 1 = 52$$

$UB_1 = 52 - min\{52, 5\} + 0 = 47$
$UB_2 = 52 - min\{52, 12\} + 6 = 46$
$UB_3 = 52 - min\{52, 19\} + 15 = 48$

$D_1 = 5 - 0 = 5$ $D_2 = 12 - 6 = 6$ $D_3 = 15 - 19 = 4$

In the previous iteration, machine 2 had the maximum load, therefore the backtracking should be started from that machine.

M2: 0(18) 2(17) 0(16) 1(10) $L_2 = 44$ $C_2 = 44 + 6 = 50$

$\frac{142 - (46 + 44) + 4}{3 - 2} = 56 > 52$ The load is infeasible, and we should proceed to lemma 4.6.4.

$\frac{142 - (47 + 46) + 4}{3 - 2} = 53 > 52$ The lemma is satisfied, and it is proved that there is no other solution for the problem.

Therefore the optimal solution for this problem is

$$C^*_{max} = UB + 1 = 52 + 1 = 53$$

The Gantt chart for the optimal solution is shown below.

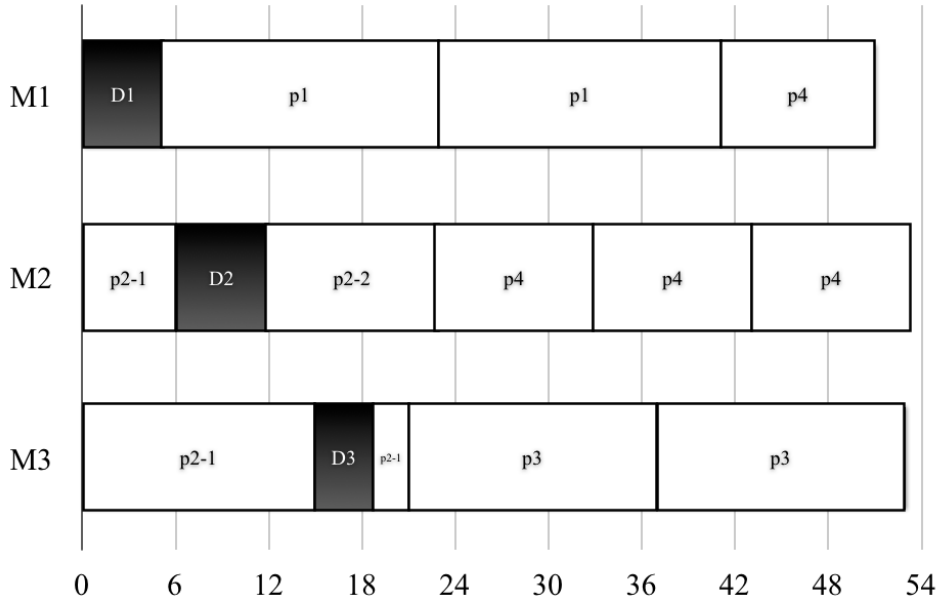

Figure 4.3: Schedule Obtained by the Exact Algorithm

Chapter 5

Experimental Results and Discussion

Three series of computational experiments have been carried out to evaluate the performance of the enumeration algorithm developed in the previous chapter. Different sets of problem instances are used in each experiment to demonstrate the effectiveness of the proposed algorithm.

In order to apply different settings, any variable that is likely to significantly impact the performance of the algorithm is included in the list of parameters to be varied. The number of jobs (N) to be scheduled is an important factor since it directly affects the load of the system. The number of machines (m) is another important factor because it affects the distribution of jobs on the machines. The third factor is the number of job types (n) or the number of different processing times. The problem becomes more challenging when the number of different processing times is small. Whenever the load on a machine is infeasible, it is always easier to find a new feasible load from a large variety of job types rather than from small one.

The experiments are designed with different levels of difficulties, therefore the classification of problems in the experiments are not equal. The difficulty of the experiments is described by the number of iterations required by the algorithm to find the optimal solution. To prevent excessive computational time, whenever a problem is not solved within a predetermined time limit, the computation is interrupted. Another reason for stopping the computations is memory overflow. In all experiments, the difficulty of the problem is increased until one of the two stopping situations is encountered.

All experiments are carried out on a personal computer (Pentium D, 2.99 GHz, 3.22 GB of RAM) in the Industrial Engineering Department at Dalhousie University.

This chapter includes the codes of the various models developed and describes the

experiments conducted. Numerical results are then presented and the efficiency of the proposed exact algorithm is shown.

5.1 Coding Method

The algorithm has been coded in C language Microsoft Visual Studio [37]. The C input model is supplied in Appendix A. A time limit of 3600 s (1h), is considered for solving the problems. The best feasible solution is provided whenever the algorithm is unable to solve the problem within this time limit.

To test the effectiveness of the new algorithm the ILP model was solved by one of the best commercial solvers available: CPLEX . For this purpose the ILP models are written into the ILOG Language and runs in ILOG CPLEX 11.2.0 [17]. CPLEX implements optimizers based on the simplex algorithms, which used primal simplex, dual simplex, barrier algorithms and sifting algorithm, when the problem contains extractable network substructure[16].

In order to prevent excessive computational time, whenever a problem is not solved by the CPLEX within the time limit of 7200s (2h), computation is stopped for the problem. At this point ILOG reports the best feasible solution, which has been found within the time limit. The ILOG codes for model 1 and 2 are supplied in Appendix B.

Except for the Graham's experiment, the parameters used in the rest of experiments are randomly generated. The random parameters are obtained by the Mersenne Twister Random Number Generator [38]. Mersenne Twister is a common random number generators, and experts consider it an excellent generator. The code of the random generator is provided in Appendix C.

5.2 Experiments

In this section, different experiments have been conducted to show the performance measure of the ILPs and the exact algorithm. In all experiments, the Central Processing Unit time (CPUt) represents the time in seconds required to find the best

feasible solution.

5.2.1 Graham's Example

The first experiment is based on the famous example given by Graham [14] for the list scheduling of $P_m \mid\mid C_{max}$. He proved that the performance ratio of this problem is at most

$$\frac{4}{3} - \frac{1}{3m}$$

and gave an example achieving this value. Later Dósa [12] proved that this example is the only one having this performance ratio and list scheduling gives better solution on any other problem instances.

As parallel machine scheduling with availability constraint problem is very closely related to the $P_m \mid\mid C_{max}$ problem, it is natural to test the new algorithm and the ILP models on that class of problems. In this experiment, different instances of the problem are generated for this problem type.

In the original example by Graham, the number of jobs is $2m + 1$ in the case of m machines. The processing times are in non-increasing order:

$$2m - 1,\ 2m - 1,\ 2m - 2,\ 2m - 2,\ \ldots,\ m + 1,\ m + 1,\ m,\ m,\ m.$$

The optimal value of the appropriate $P_m \mid\mid C_{\max}$ problem is $3m$ and traditionally the optimal solution is written is the form:

$$\begin{aligned} &\text{Machine 1: } 2m - 1,\ m + 1 \\ &\text{Machine 2: } 2m - 1,\ m + 1 \\ &\text{Machine 3: } 2m - 2,\ m + 2 \\ &\qquad \ldots \\ &\text{Machine m: } m,\ m,\ m \end{aligned} \tag{5.1}$$

Note that this solution is true, when there is no availability constraint on the machines.

Series 1. Single Unavailability

In this experiment, the number of availability constraint on each machine is limited to one. The other data for this experiment are shown in the table below. In the original Graham's experiment non-availability periods were not considered. Therefore, a modified version of Graham's experiment is used.

Table 5.1: Parameters for the Graham Experiment

Number of Jobs	$2m+1$
Number of Machines	m
Processing Time of Jobs	$2m-1, 2m-1, 2m-2, 2m-2, \ldots, m+1$ $, m+1, m, m, m$
Number of Unavailabilities	m
Duration of Unavailabilities	15

The non-availability periods start at instant 0 for machines with odd index and at instant 15 for machines with even index.

Table 5.2 displays the result of the modified Graham's experiment for the exact algorithm.

Table 5.2: Results for the Modified Graham's Example Series 1

m	N	C^{LPT2}	C^{Alg}	Optimal Solution?	CPUt
100	201	414	315	Yes	6
200	401	814	615	Yes	11
300	601	1214	915	Yes	46
400	801	1614	1215	Yes	144
500	1001	2014	1515	Yes	348
600	1201	2414	1815	Yes	720
700	1401	2814	2115	Yes	1327
800	1601	3214	2415	Yes	1327
900	1801	3614	2715	Yes	3621

The same experiment is applied to the ILP model 1. The value of M has to be defined and selected appropriately to solve the ILP. This value should be large enough to allow the model to find feasible solutions, but be moderate to avoid excessive

computations analysis. For that reason the value of M is set to the optimal solution which is found by the exact algorithm for the problem.

Table 5.3 depicts the result of this experiment. For each instance (m, N) of the problem, table 5.3 gives the solution obtained by LPT2, the solution reached by the exact algorithm, the CPUt of the exact algorithm and shows if the solution by the algorithm is optimal or not. The reason of termination for the problems without the optimal solution is memory overflow.

Table 5.3: Computational Experiments for the ILP with Graham's Example Series 1

m	N	[a] C^{ILP}	[b] C^{alg}	ILP Optimal?	[c] $CPUt^{ILP}$	[d] $CPUt^{Alg}$	m	N	C^{ILP}	C^{alg}	ILP Optimal?	$CPUt^{ILP}$	$CPUt^{Alg}$
30	**61**	105	105	Yes	8	< 1	**56**	**113**	187	183	No	429	< 1
31	**63**	108	108	Yes	38	< 1	**57**	**115**	189	186	No	712	< 1
32	**65**	111	111	Yes	20	< 1	**58**	**117**	194	189	No	381	< 1
33	**67**	114	114	Yes	16	< 1	**59**	**119**	189	192	No	561	< 1
34	**69**	117	117	Yes	27	< 1	**60**	**121**	200	195	No	448	< 1
35	**71**	120	120	Yes	13	< 1	**61**	**123**	203	198	No	445	< 1
36	**73**	123	123	Yes	11	< 1	**62**	**125**	207	201	No	666	< 1
37	**75**	126	126	Yes	80	< 1	**63**	**127**	209	204	No	617	< 1
38	**77**	129	129	Yes	150	< 1	**64**	**129**	214	207	No	435	< 1
39	**79**	132	132	Yes	82	< 1	**65**	**131**	210	210	Yes	446	< 1
40	**81**	135	135	Yes	15	< 1	**66**	**133**	220	213	No	557	< 1
41	**83**	138	138	Yes	302	< 1	**67**	**135**	225	216	No	606	< 1
42	**85**	141	141	Yes	298	< 1	**68**	**137**	223	219	No	863	< 1
43	**87**	144	144	Yes	46	< 1	**69**	**139**	227	222	No	663	< 1
44	**89**	147	147	Yes	233	< 1	**70**	**141**	232	225	No	709	< 1
45	**91**	150	150	Yes	28	< 1	**71**	**143**	231	228	No	965	< 1
46	**93**	153	153	Yes	56	< 1	**72**	**145**	235	231	No	604	< 1
47	**95**	156	156	Yes	55	< 1	**73**	**147**	241	234	No	667	< 1
48	**97**	159	159	Yes	728	< 1	**74**	**149**	242	237	No	839	< 1
49	**99**	162	162	Yes	128	< 1	**75**	**151**	249	240	No	793	< 1
50	**101**	165	165	Yes	46	< 1	**76**	**153**	247	243	No	700	< 1
51	**103**	169	168	No	3,094	< 1	**77**	**155**	250	246	No	657	< 1
52	**105**	171	171	Yes	72	< 1	**78**	**157**	252	249	No	941	< 1
53	**107**	176	174	No	566	< 1	**79**	**159**	259	252	No	809	< 1
54	**109**	180	177	No	569	< 1	**80**	**161**	250	255	No	862	< 1
55	**111**	184	180	No	567	< 1							

a makespan for ILP
b makespan for Exact Algorithm
c CPUt for ILP
d CPUt for ILP

According to the results in table 5.3, the exact algorithm outperforms the ILP in the current version of CPLEX. The ILP can optimally solve problems with up to 50 machines and 101 jobs. After that, the ILP has found optimal solution for two more problems, and for the rest of problems a quasi-optimal solution has been reported within the time limit. As can be seen from the table, the CPUt for the ILP has never reached the time limit, meaning that the program exceeds the memory before it can reach the time limit or find the optimal solution.

Series 1. Multiple Availability

The second series of this experiment investigates problems with two availability constraints on the machines. The non-availability periods are defined as follows with duration 15:

Assume i is an odd number then

$$s_{i,1} = 0 \qquad s_{i,2} = 30$$
$$s_{i+1,1} = 15 \qquad s_{i+1,2} = 45$$

The rest of parameters can be found from table 5.1. Table 5.4 shows the results of this experiment.

Table 5.4: Computation Experiments with Graham's Example Series 2

m	n	C^{LPT2}	C^{Alg}	Optimal Solution?	C^{LPT2}/C^{Alg}	CPUt (sec)
100	201	429	330	Yes	1.30	0
200	401	829	630	Yes	1.32	9
300	601	1229	930	Yes	1.32	48
400	801	1629	1230	Yes	1.32	146
500	1001	2029	1530	Yes	1.33	356
600	1201	2429	1830	Yes	1.33	734
700	1401	2829	2130	Yes	1.33	1352
800	1601	3598	2430	Yes	1.48	3193

The exact algorithm can schedule 1601 jobs on 800 machines within the time limit. If the number of jobs exceeds 1601 then the CPUt exceeds 3600 seconds.

The same experiment is run for the ILP. Table 5.5 shows the results of this experiment for both ILP and exact algorithm. The exact algorithm has solved all the problems optimally in less than a second (< 1).

Table 5.5: Computation Experiments for ILP with Graham's Example Series 2

m	N	C^{ILP}	C^{alg}	ILP Optimal?	$CPUt^{ILP}$	$CPUt^{Alg}$	m	N	C^{ILP}	C^{alg}	ILP Optimal?	$CPUt^{ILP}$	$CPUt^{Alg}$
30	**61**	120	120	Yes	11	< 1	**56**	**113**	202	198	No	427	< 1
31	**63**	125	123	No	138	< 1	**57**	**115**	207	201	No	542	< 1
32	**65**	126	126	Yes	110	< 1	**58**	**117**	211	204	No	479	< 1
33	**67**	129	129	Yes	29	< 1	**59**	**119**	215	207	No	419	< 1
34	**69**	132	132	Yes	129	< 1	**60**	**121**	215	210	No	605	< 1
35	**71**	135	135	Yes	113	< 1	**61**	**123**	223	213	No	525	< 1
36	**73**	142	138	No	336	< 1	**62**	**125**	223	216	No	589	< 1
37	**75**	144	141	No	260	< 1	**63**	**127**	224	219	No	551	< 1
38	**77**	146	144	No	304	< 1	**64**	**129**	229	222	No	598	< 1
39	**79**	147	147	Yes	28	< 1	**65**	**131**	228	225	No	624	< 1
40	**81**	153	150	No	270	< 1	**66**	**133**	233	228	No	558	< 1
41	**83**	156	153	No	209	< 1	**67**	**135**	236	231	No	652	< 1
42	**85**	159	156	No	253	< 1	**68**	**137**	241	234	No	683	< 1
43	**87**	161	159	No	530	< 1	**69**	**139**	244	237	No	574	< 1
44	**89**	165	162	No	315	< 1	**70**	**141**	247	240	No	638	< 1
45	**91**	165	165	Yes	1,139	< 1	**71**	**143**	247	243	No	654	< 1
46	**93**	171	168	No	339	< 1	**72**	**145**	253	246	No	540	< 1
47	**95**	174	171	No	427	< 1	**73**	**147**	258	249	No	744	< 1
48	**97**	174	174	Yes	4,557	< 1	**74**	**149**	263	252	No	950	< 1
49	**99**	181	177	No	478	< 1	**75**	**151**	262	255	No	908	< 1
50	**101**	183	180	No	375	< 1	**76**	**153**	265	258	No	694	< 1
51	**103**	185	183	No	405	< 1	**77**	**155**	274	261	No	594	< 1
52	**105**	193	186	No	275	< 1	**78**	**157**	271	264	No	728	< 1
53	**107**	191	189	No	546	< 1	**79**	**159**	276	267	No	746	< 1
54	**109**	195	192	No	392	< 1	**80**	**161**	277	270	No	662	< 1
55	**111**	200	195	No	339	< 1							

5.2.2 Problems with Random Uniform Processing Times Between 1 and 99

In this experiment processing times are random positive integer numbers drawn from the uniform distribution $U(1, 99)$. Although many experiments have been carried out, only the results of the largest problems are reported here. For each problem size, 10 different problems are generated. The minimum, maximum and average CPUt of these 10 instances are reported. The same is true for the series of experiments discussed in the next subsection.

Series No. 1

In this series the number of machines is fixed to $m = 3$, and the number of jobs are increased until either the program is out of the memory, or CPUt exceeds the time-limit. The input data for this experiment are shown in table below.

Table 5.6: Parameters for $U(1, 99)$ for Series No. 1

Number of Machines	3
Number of Jobs	N
Processing Time of Jobs	$U(1, 99)$
Number of Availability constraintt on each Machine	1
Duration of Unavailabilities	15
Start of Unavailability Period on i^{th} Machine	$15(i-1)$

The table in the next page represents the results for this experiment. As can be seen from the table 5.7 the makespan of LPT2 algorithm and exact algorithm are equal. The reason is that, list scheduling results has a high performance, when number of jobs is large and are picked uniformly between large interval of time. However, the same statement is not true when the jobs are distributed non-uniformly, as it is shown in Graham's experiment.

Our program was able to solve problems up to 1,000,000 jobs. Above this limit the program does not have enough memory to solve the problem. The short CPUt

can be explained by the fact that the depth of the enumeration, i.e. the depth of the stack, is never more than 3.

Table 5.7: Results for Random Problems having $U(1, 99)$ Processing Times and $m = 3$

$\mathbf{m = 3; N = 800,000}$					$\mathbf{m = 3; N = 900,000}$					$\mathbf{m = 3; N = 1,000,000}$				
Runs	C^{LPT2}	C^{Alg}	Optimal Solution?	CPUt	Runs	C^{LPT2}	C^{Alg}	Optimal Solution?	CPUt	Runs	C^{LPT2}	C^{Alg}	Optimal Solution?	CPUt
1	13338333	13338333	Yes	2	**1**	15001330	15001330	Yes	2	**1**	16675435	16675435	Yes	2
2	13330728	13330728	Yes	2	**2**	15009225	15009225	Yes	2	**2**	16661686	16661686	Yes	2
3	13342017	13342017	Yes	2	**3**	15000131	15000131	Yes	2	**3**	16664161	16664161	Yes	2
4	13338287	13338287	Yes	2	**4**	14992145	14992145	Yes	2	**4**	16653583	16653583	Yes	2
5	13331792	13331792	Yes	2	**5**	14992094	14992094	Yes	2	**5**	16678583	16678583	Yes	2
6	13328090	13328090	Yes	2	**6**	14993070	14993070	Yes	2	**6**	16662365	16662365	Yes	2
7	13341988	13341988	Yes	2	**7**	14978187	14978187	Yes	2	**7**	16666658	16666658	Yes	2
8	13330333	13330333	Yes	2	**8**	15000596	15000596	Yes	2	**8**	16665268	16665268	Yes	2
9	13321950	13321950	Yes	2	**9**	14990542	14990542	Yes	2	**9**	16678039	16678039	Yes	2
10	13322636	13322636	Yes	2	**10**	14992911	14992911	Yes	2	**10**	16667824	16667824	Yes	2
			Min	**2**				**Min**	**2**				**Min**	**2**
			Max	**2**				**Max**	**2**				**Max**	**2**
			Average	**2**				**Average**	**2**				**Average**	**2**

Series No. 2

In this series, the ratio of the number of machines over the number of jobs is fixed ($N = m^2$). Table 5.8 summarizes the parameters used for this experiment.

Table 5.8: Parameters for $U(1, 99)$ for Series No. 2

Number of Machines	m
Number of Jobs	m^2
Processing Time of Jobs	$U(1, 99)$
Number of Availability Constraint on Each Machine	1
Duration of Unavailabilities	15
Start of Unavailability period on i^{th} Machine	$15(i-1)$

Here the depth of the enumeration is much greater than in the previous case. But still the exact algorithm was able to solve problems with $250,000$ jobs. After that the program does not have sufficient memory to solve the problem. The result of this experiment is shown on the following page.

Table 5.9: Results for Random Problems having $U(1, 99)$ Processing Times and $N = m^2$

$m = 300; N = 90,000$					$m = 400; N = 160,000$					$m = 500; N = 250,000$				
Run	C^{LPT2}	C^{Alg}	Optimal Solution?	CPUt	Run	C^{LPT2}	C^{Alg}	Optimal Solution?	CPUt	Run	C^{LPT2}	C^{Alg}	Optimal Solution?	CPUt
1	15023	15023	Yes	32	**1**	2002	2002	Yes	73	**1**	25009	25009	Yes	142
2	15020	15020	Yes	31	**2**	20021	20021	Yes	73	**2**	24977	24977	Yes	142
3	15047	15047	Yes	31	**3**	20027	20027	Yes	73	**3**	25007	25007	Yes	142
4	15028	15028	Yes	31	**4**	20034	20034	Yes	73	**4**	25011	25011	Yes	142
5	15069	15069	Yes	31	**5**	20026	20026	Yes	73	**5**	24982	24982	Yes	142
6	15009	15009	Yes	31	**6**	19984	19984	Yes	73	**6**	25023	25023	Yes	142
7	15003	15003	Yes	31	**7**	20045	20045	Yes	73	**7**	24966	24966	Yes	142
8	14989	14989	Yes	31	**8**	20027	20027	Yes	73	**8**	25004	25004	Yes	142
9	15038	15038	Yes	31	**9**	20021	20021	Yes	73	**9**	24980	24980	Yes	142
10	15019	15019	Yes	31	**10**	19993	19993	Yes	73	**10**	25042	25042	Yes	142
			Min	**31**				**Min**	**73**				**Min**	**142**
			Max	**32**				**Max**	**73**				**Max**	**142**
			Average	**31**				**Average**	**73**				**Average**	**142**

5.2.3 Problems with Random Uniform Processing Times Between 5 and 15

In this experiment, the processing times are random positive integer numbers drawn from the uniform distribution $U(5, 15)$. Two similar series of experiences have been carried out. This experiment is slightly more difficult than the problem with $U(1, 99)$ bcause of the variety of processing times. It is always faster to find a feasible load for an infeasible machine, when there is the choice of picking a job from a batch of 100 different processing times than in a batch of 10.

Here problems with very large number of jobs have been solved. On the other hand this is the only case when not all of the problems having a large size were solved until optimally within the time limit of 3600 seconds. However for those problems, the exact algorithm has provided the quasi-optimal solution.

Series No. 1

In three cases, the algorithm was unable to find the optimal solution or perhaps prove the optimality of the feasible solution within the time limit. In all of the three cases, the objective function value of the best known feasible solution was greater than the lower bound by 1, which shows the high possibly of the optimality of the feasible solution.

The following table shows the results of this experiment. Problems with up to 1,000,000 jobs are solved optimally. For larger problems, memory overflows. CPUt is reported for the instances that are optimally solved.

Table 5.10: Results for Random Problems having $U(5, 15)$ Processing Times and $m = 3$

$\mathbf{m = 3; N = 800,000}$					$\mathbf{m = 3; N = 900,000}$					$\mathbf{m = 3; N = 1,000,000}$				
Runs	C^{LPT2}	C^{Alg}	Optimal Solution?	CPUt	Runs	C^{LPT2}	C^{Alg}	Optimal Solution?	CPUt	Runs	C^{LPT2}	C^{Alg}	Optimal Solution?	CPUt
1	2997534	2997534	Yes	< 1	**1**	3000361	3000361	Yes	< 1	**1**	3335608	3335606	Yes	< 1
2	2998542	2998542	Yes	< 1	**2**	3000755	3000755	Yes	< 1	**2**	3334270	3334268	Yes	< 1
3	2999300	2999300	Yes	< 1	**3**	2998183	2998183	Yes	< 1	**3**	3334449	3334446	Yes	< 1
4	2999046	2999046	Yes	< 1	**4**	2999472	2999472	Yes	< 1	**4**	3333999	3333997	No	—
5	3001724	3001724	Yes	< 1	**5**	3000309	3000309	Yes	< 1	**5**	3333595	3333592	Yes	< 1
6	2999309	2999309	No	—	**6**	3000176	3000176	Yes	< 1	**6**	3332189	3332186	Yes	< 1
7	3000313	3000313	Yes	< 1	**7**	3000476	3000476	Yes	< 1	**7**	3335298	3335295	Yes	< 1
8	2999432	2999432	Yes	< 1	**8**	3000586	3000586	No	—	**8**	3334953	3334951	Yes	< 1
9	2999697	2999697	Yes	< 1	**9**	2999701	2999701	Yes	< 1	**9**	3332040	3332037	Yes	< 1
10	2998632	2998632	Yes	< 1	**10**	2998540	2998540	Yes	< 1	**10**	3334449	3334446	Yes	< 1
			Min	< 1				**Min**	< 1				**Min**	< 1
			Max	< 1				**Max**	< 1				**Max**	< 1
			Average	< 1				**Average**	< 1				**Average**	< 1

series No. 2

In this series, the ratio of the number of machines over the number of jobs is fixed as in the previous case ($N = m^2$). The processing times are drawn uniformly between 5 and 15. The other parameters are the same as in table 5.8.

Table 5.11 gives the result of this experiment. Although the exact algorithm was unable to find the optimal solution for some problems, in most cases the initial solution that provided by LPT2 is improved. For $N = 90000$, nine out of 10 problems were solved immediately, i.e. 10 seconds CPUt was observed. For the other problem the initial solution provided by LPT2 is improved by one by the exact algorithm. For the case of $N = 160000$, eight out of 10 problems were solved. The quasi-optimal solution of further two problems is greater than the lower bound by 1. For $N = 250000$, six out of 10 problems were solved optimally.

Table 5.11: Results for Experiment Series 4.2

$m = 300; N = 90,000$					$m = 400; N = 160,000$					$m = 500; N = 250,000$				
Runs	C^{LPT2}	C^{Alg}	Optimal Solution?	CPUt	Runs	C^{LPT2}	C^{Alg}	Optimal Solution?	CPUt	Runs	C^{LPT2}	C^{Alg}	Optimal Solution?	CPUt
1	3011	3009	Yes	11	**1**	4013	4011	Yes	24	**1**	5017	5015	Yes	46
2	3013	3011	Yes	10	**2**	4017	4016	Yes	24	**2**	5017	5015	Yes	54
3	3008	3006	Yes	10	**3**	4015	4013	Yes	28	**3**	5013	5012	Yes	—
4	3010	3008	Yes	10	**4**	4012	4011	Yes	23	**4**	5013	5011	Yes	46
5	3010	3008	Yes	11	**5**	4008	4007	No	—	**5**	5013	5013	No	—
6	3011	3009	Yes	11	**6**	4013	4011	Yes	31	**6**	5008	5007	Yes	46
7	3014	30012	Yes	10	**7**	4015	4013	Yes	23	**7**	5013	5011	Yes	46
8	3013	3011	Yes	10	**8**	4014	4012	Yes	24	**8**	5012	5010	Yes	46
9	3010	3007	Yes	10	**9**	4012	4010	Yes	23	**9**	5010	5009	No	—
10	3018	3017	No	—	**10**	4015	4015	No	—	**10**	5014	5012	No	—
			Min	**10**				**Min**	**23**				**Min**	**46**
			Max	**11**				**Max**	**31**				**Max**	**54**
			Average	**10**				**Average**	**25**				**Average**	**47**

A summary of the results of the experiments generated by the random numbers can be found in the table below.

Table 5.12: Summary of the Results

				CPUt		
m	N	n	Processing Time	min	max	Average CPUt
3	800,000	16	$U(5,15)$	< 1	< 1	< 1
3	800,000	99	$U(1,99)$	2	2	2
3	900,000	16	$U(5,15)$	< 1	< 1	< 1
3	900,000	99	$U(1,99)$	2	2	2
3	1,000,000	16	$U(5,15)$	< 1	< 1	< 1
3	1,000,000	99	$U(1,99)$	2	2	2
300	90,000	16	$U(5,15)$	10	11	10
300	90,000	99	$U(1,99)$	31	32	31
400	160,000	16	$U(5,15)$	23	31	25
400	160,000	99	$U(1,99)$	73	73	73
500	250,000	16	$U(5,15)$	46	54	47
500	250,000	99	$U(1,99)$	142	142	142

Chapter 6

Conclusion and Future Studies

In the literature, most scheduling problems have been solved by assuming that all machines are continuously available, which obviously is not always the case in practice.

In this study we have considered the problem of parallel machine scheduling with multiple planned unavailability periods in the resumable case. We have studied the minimization of the makespan ($P_m \mid r - a_{m,q} \mid C_{max}$).

The problem has been formulated as a mathematical programming problem. An effective algorithm has been developed to solve large-scale and practical problems. The algorithm loads machines according to the lexicographical order within a construction and backtracking approach. LPT2 algorithm [25] is employed to generate an initial upper bound for the exact algorithm.

A large set of numerical experiments was carried out. The results demonstrate that the exact algorithm is able to solve large-scale problems which are not solvable by any other method including the current best ILP solver. Based on the results of the experiments for the ILP, we demonstrated that the proposed ILP can be used to effectively solve the small-size problems. The performance of the proposed algorithm is independent of the pattern of non-availability periods.

In the literature there is a limited number of algorithms which can solve optimally this type of problem. As it was mentioned before these algorithms can only solve problems with two machines and single availability constraint, or very small size problems. In general, these algorithms cannot schedule more than 100 jobs. The other algorithms in the literature either result in non-optimal solution, cannot apply to large-scale problems, or if they can, they do not find a solution in a reasonable time.

This study can, in the future, be extended in multiple directions. Further research

could be directed to the development of new methods for scheduling jobs with non-resumable or semi-resumable jobs. Another interesting line of research is the joint scheduling of both unavailability periods and jobs on the machines. The study can also be extended to more difficult scheduling problems such as flow shop or flexible manufacturing systems. Different performance measures such as due-date related objective function or weighted completion time as well as problems with stochastic processing times and machine breakdowns, activity modifying-rate maintenance and machine eligibility constraints can be studied.

All the codes used in this thesis, as well as the sets of experiments are available upon request from the author.

Bibliography

[1] Adiri, I., Bruno, J., Frostig, E. & Rinnooy Kan, A.H.G., 1989, Single Machine Flow-Time Scheduling with a Single Breakdown, *Acta Informatica*, Vol 26, pp.679-696.

[2] Anglia Ruskin University, 2008. *Guide to the Harvard Style of Referencing.* [Online] (Updated 26 May 2009)
Available at:http://libweb.anglia.ac.uk/referencing/harvard.htm
[Accessed 10 April 2005]

[3] Baker, K.R. & Trietsch, D., 2009, *Principles of Sequencing and Scheduling*, Wiley, Hoboken, NJ.

[4] Błażwicz, J., Maciej, D., Formanowicz, P., Kubiak, W. & Schmidt, G., 2000, Scheduling Preemptable Tasks on Parallel Processors with Limited Availability , *Parallel Computing*, Vol 26, pp.1195-1211.

[5] Błażwicz, J., Breit, J., Formanowicz, P., Kubiak, W. & Schmidt, G., 2001, Heuristic Algorithms for the Two-Machine Flowshop with Limited Machine Availability, *Omega*, Vol 29, pp.559-608.

[6] Brucker, P., 2007, *Scheduling Algorithms*, Springer, Verlag Berlin Heidelberg .

[7] Cassady.R.C. & Kutanoglu, E, 2003, Minimizing job tardiness using integrated preventive maintenance planning and production scheduling, *IEEE Transactions on Reliability*, Vol. 35, pp.503-513.

[8] Cassady.R.C. & Kutanoglu, E, 2005, Integrating Preventive Maintenance Planning and Production Scheduling for a Single Machine, *IEEE Transactions on Reliability*, Vol. 54, pp.304-309.

[9] Coffman, E.G., 1976, Introduction to Deterministic Scheduling Theory: In E.G. Coffman, ed. *Computer and Job-Shop Scheduling Theory.* John Wiley & Sons, Inc. Ch. 1.

[10] Coffman, E.G., Garey, M.R. & Johnson, D.S., 1978, An Application of Bin-Packing to Multiprocedure Scheduling, *SIAM Journal of Computing*, Vol 7, pp.1-17.

[11] Deitel, H.M., & Deitel, P.J., 2001. How to Program C. 3^{rd} ed. New Jersey: Prentice Hall.

[12] Dósa, G.T., 2004, Graham's Example is the Only Tight One for $P \mid\mid C_{\max}$, *Annales Universitatis Scientarium Budapestiensis de Rolando Eötvös nominatae, Sectio Mathematica*, Vol 47, pp.207-210.

[13] Graham, R.L., 1966, Bounds for Certain Multiprocess Anomalies ,*Bell System Technology Journal*, Vol 45, pp.1563-1581.

[14] Graham, R.L., 1969, Bounds on Multiprocessing Timing Anomalies,*SIAM Journal of Applied Mathematics*, Vol. 17, pp.416-429.

[15] Graham, R.L., Lawer, E.L., Lenstra, J.K. & Rinnooy Ken, A.H.G., 1979, Optimization and Approximation in Deterministic Sequencing and Scheduling,*Annals of Discrete Mathematics*, Vol. 5, pp.287-326.

[16] Hentenryck, P. V, 1999, *The OPL Optimization Programming Language*, Massachusetts Institute of Technology.

[17] ILOG OPL Development Studio Software. (2008). ILOG CPLEX 11.2.0.

[18] Hashemian, N. & Vizvári, B., 2008. *A Method to Schedule Both Transportation and Production at the Same Time in a Special FMS.* [Online] New Jersey: RUTCOR.
Available at:http://rutcor.rutgers.edu/pub/rrr/reports2008/12_08.pdf
[Accessed 12 April 2008]

[19] Ho, J.c. & Wong J.S., 1995. Makespan minimization for m parallel identical processors *Naval Research Logistics*, Vol. 42, pp.935-948.

[20] Kellereer, H., 1998, Algorithms for Multiprocessor Scheduling with Machine Release Times,*IIE Transactions*, Vol. 30, pp.991-999.

[21] Kopka, H. & Daly, P. W., 2004, *A Guide to Latex and Electronic Publishing*, 4^{th} ed. Essex: Addison Wesley Longman Limited.

[22] Labetoulle, J., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., 1979, Preemptive Scheduling of Uniform Machines Subjected to Due Dates, *Technical Paper BW 99/79*, CWI, Amsterdam.

[23] Lee. C.Y., Massey, J. D., 1988, Multiprocess Scheduling: An Extension of the Multifit Algorithm, *Journal of Manufacturing Systems*, Vol. 7, pp.25-32.

[24] Lee. C.Y., 1991, Parallel Machines Scheduling with Nonsimultaneous Machine Available Time, *Discrete Applied Mathematics*, Vol. 30, pp.53-61.

[25] Lee. C.Y., 1996, Machine Scheduling with an Availability Constraint, *Journal of Global Optimization*, Vol. 9, pp.395-416.

[26] Lee. C.Y., 1999, Two-Machine Flowshop Scheduling with availability constraints, *European Journal of Operation Research*, Vol. 114, pp.420-429.

[27] Lee. C.Y., 2000, A Note on "Parallel Machine Scheduling with Non-Simultaneous Machine Available Time", *Discrete Applied Mathematics*, Vol. 100, pp.133-135.

[28] Lee, C.Y. & Liman, S.D, 1992, Single Machine Flow-Time Scheduling with Scheduled Maintenance., *Acta Informatica*, Vol. 29, pp.375-382.

[29] Lenstra, J.K., Rinnooy Kan, A.H.G., Brucker, P., 1977, Complexity of Machine Scheduling Problems, *Annals of Discrete Mathematics*, Vol. 1, pp.342-362.

[30] Leung. J.Y.T, 2004, *Handbook of Scheduling, Algorithms, Models, and Performance Analysis*, CHAPMAN & HALL/CRC, United States of America.

[31] Liao, C.J., Shyur, D.L., Lin, C.H., 2005, Makespan Minimization for Two Parallel Machines with an Availability Constraint, *European Journal of Operation Research*, Vol. 160, pp.445-456.

[32] Liao, L.W., Sheen, G.J., 2008, Parallel Machine Scheduling with Machine Availability and Eligibility Constraints, *European Journal of Operation Research*, Vol. 184, pp.458-467.

[33] Lin, G., He, Y., Yao, Y., Lu, H., 1997, Exact bounds of the modified LPT Algorithms Applying to Parallel Machines Scheduling with Nonsimultaneous Machine Available Times, *Applied Mathematics-A Journal of Chinese Universities*, Vol. 12, pp.109-116

[34] Lin, C.H., Liao, C.J., 2007, Makespan Minimization for Two Parallel Machines with an Unavailable Period on Each Machine, *International Journal of Advance Manufacturing Technology*, Vol. 33, pp.1024-1030.

[35] Liu, Z. & Sanlaville, E., 1995, Preemptive Scheduling with Variable Profile, Precedence Constraints and Due Dates, *Disc. Appl. Maths.*, Vol. 58, pp.253-280.

[36] Ma, Y., Chengbin, C. & Chunrong, Z., 2010, A Survey of Scheduling with Deterministic Machine Availability Constraints, *Computers & Industrial Engineering*, Vol. 50, pp.199-211.

[37] Microsoft Visual Studio Software. (2008). C++ 9.0.

[38] Matsumoto, M.& Nishimura, T., 1998, Mersenne Twister, *ACM Transactions on Modeling and Computer Simulation*, Vol. 8, pp.3-30.

[39] Muntz, R.& Coffman, E.G., 1970, Preemptive Scheduling of Real-Time Tasks on Multiprocessor Systems, *Journal of the ACM*, Vol. 17, pp.324-338.

[40] Sanlaville, E., 1995, Nearly On Line Scheduling of Preemptive Independent Tasks, *Discrete Applied Mathematics*, Vol. 57, pp.229-241.

[41] Schmidt, G., 1984, Scheduling on Semi-Identical Processors, *Z. Opns Res.*, Vol. A28, pp.153-162.

[42] Schmidt, G., 1998, Scheduling with Limited Machine Availability, *European Journal of Operational Research*, Vol. 121, pp.1-15.

[43] Sleator, D.D., Tarjan, R.E, 1985, Amortized Efficiency of the List Update and Paging Rules, *Communication of the ACM*, Vol. 28, pp.202-208.

[44] Tan, Z., Yong, H., 2002, Optimal Online Algorithm for Scheduling on Two Identical Machines with Machine Availability Constraints , *Information Processing Letters*, Vol. 83, pp.323-329.

[45] Wu, C.W., Lee. W. C., 2003, Scheduling Linear Deteriorating Jobs to Minimize Makespan with an Availability Constraint on a Single Machine, *Information Processing Letters*, Vol. 87, pp.89-93.

[46] Wu, C.W., Lee, W.C., 2007, A Note on Single-Machine Scheduling with Learning Effect and an Availability Constraint, *The International Journal of Advanced Manufacturing Technology*, Vol. 33, pp.540-544.

Appendices

Appendix A

C Input Model for Exact Algorithm

```
/*****************************************
 * Exact Algorithm Model (Moharejeh)
 * Author: Navid Hashemian
 * Creation Date: July 8, 2009
 *****************************************/

#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "randomc.h"
#include "mersenne.cpp"

#define Mlimit 910
//Limit for Number of Machines
#define JobTypeLimit 9000
// Limit for Number of JobTypes
#define Alimit 500
//Limit for Availability Constraint
#define minimum 100000000
//A Constant

void InitialAvailability (void) ;
//LPT2_1
void Lemma_1 (void) ;
//Calculating Upper Bound of the Machines
void Lemma_2 (void) ;
//Calculating Duration of Unavailability Periods for the Machines
void LPT (void);
//LPT2_2
void LPT2 (void) ;
//LPT2_3
void LoadingB (void);
//To Schedule the Next Lexicographical Load on Machine i
void Maximum (void);
//For Calculating the Maximum Load on the Machines
void MinimumLoad (void) ;
//For Calculating the Minimum Load on the Machines
void Print (void);
//To Print the  Solution
void ProcessSwap (void);
 //Lemma 4.6.6.
void Restore (void);
//Determined the Number of Unscheduled Jobs
void Restore_P2 (void);
//Determined the Number of Unscheduled Jobs
void Theorem_1 (void);
// Lemma 4.6.1.
void Theorem_2 (void);
//Lemma 4.6.3.
void Theorem_4 (void) ;
//Lemma 4.6.4.
```

```
void Theorem_5 (void)  ;
//Lemma 4.6.5.-Lemma 4.6.6 Part 1
void Theorem_5_P2 (void);
//Lemma 4.6.5.-Lemma 4.6.6 Part 2
void Theorem_5_P3 (void);
//Lemma 4.6.5.-Lemma 4.6.6 Part 3

clock_t c0, c1;

double cputime;


float TH_1, TH_2, TH_4;

int     AVlowerLimit, AVupperLimit, condition_1,
        condition_Availability,
        ENA[Mlimit][Alimit],
        //End of non-availability Period q of Machine i
        Flag_1, Flag_2, Flag_3,  Flag_Av[Alimit][Mlimit],
        Flag_LPT, Flag_LPT2, fr[JobTypeLimit],Flag_q ,
        Flag_Th5,
        Flag_Th5_2,
        /*If it is equal 1 then all the load of machine m is
        infeasible and backtrack step should be applied on
        machine m-2*/
        Flag_Th5_3,
        i,//index for machine
        InputCondition,
        /*If it is zero then the input data are generated
        randomly; if it is equal 1, then the program reads
        the input data from the file.*/
        j /*index for job*/, Job_Th5, Job_Th5_Back,
        JobType/*Number of Job Type*/, k,  LB/*Lower-Bound*/,
        Lmachine[Mlimit]/*Load for machine m*/,
        m/*index for machine*/, Load_Th5, Machine_Th5,
        Machine_Th5_Back,
        max,
        //The Index for the Machine with the Maximum Load
        min,
        //The Index for the Machine with the Minimum Load
        NumA[Mlimit],
        //Number of Availability Constraint for Machine i
        N[Mlimit],
        //Duration of Unavailability Periods on Machine i
        Nmach /*Number of Machines*/, Nmach_Ava,
        Nmach_Back, NumOfJobs,
        Process[JobTypeLimit]
        //Processing Time of Job Type j
        ProcessLowerLimit,
        //Lower Limit for Generating Random Processing Time
        ,ProcessUpperLimit
        //Upper Limit for Generating Random Processing Time
        , q, qStar, r, s, Q,
        Sjob
        //Smallest Scheduled Job on Machine i
        ,SNA[Mlimit][Alimit]
        //Start of Unavailability Period k on Machine i
        , SNAupperLimit, Sjob_Back,
        Sum_A
         // The Load for Machine i
        ,Sum_AV
        //The Makespan for Machine i
        ,Sum_B
        //The Load for Machine i
```

```
        ,Sum_LB, Sum_N, sum_S,
        sum_T
        //Summation of all Processing Times
        ,Sum_UBback, sum_v, UB/*Upper-Bound*/, UB_2, UB_AV,
        UB_Back[Mlimit],
        UB_Fsolution
        //To Record the Makespan for the Feasible Solution
        ,UB_LPT
        //The Upper Bound for LPT2
        v[JobTypeLimit]/*Orginal Unschedule Jobs*/,
        /*Total Number of Scheduled and Unscheduled of Job
        Type j*/
        ,y[Mlimit][JobTypeLimit]
        //Number of Schedule job from Job type i*/,
        ,z[JobTypeLimit][Mlimit]
        To Record the Solution of the Feasible Solution;

long int ir;

FILE *cfPtr;
FILE*cfPtr2;

int main()
{
        long int seed = time(0);
        TRandomMersenne rg(seed);
        InputCondition = 1;

/*********Generating Random Numbers*********/
        if ( InputCondition == 0 ){
                if ( ( cfPtr = fopen ("input.txt", "r" ) ) == NULL)
                        printf( "File couldnot be opened\n");
                else{
                        fscanf(cfPtr, "%d", &Nmach);
                        fscanf(cfPtr,"%d", &NumOfJobs);
                        fscanf(cfPtr,"%d", &ProcessLowerLimit) ;
                        fscanf(cfPtr,"%d", &ProcessUpperLimit) ;
                        for ( i=ProcessLowerLimit; i<=ProcessUpperLimit; i++ )
                                fr[i]=0;
                        for (i=1; i<=NumOfJobs; i++) {
                                ir = rg.IRandom(ProcessLowerLimit,ProcessUpperLimit);
                                //printf ("%6li   ", ir);
                                fr[ir] = fr[ir] + 1 ;
                        }
                        j=0;
                        for ( i=ProcessUpperLimit; i>=ProcessLowerLimit; i-- ){
                                if (fr[i] != 0){
                                        j = j + 1;
                                        y[0][j] = fr[i] ;
                                        Process[j] = i;
                                }
                        }
                        for (i = 1; i<=Nmach; i++)
                                fscanf(cfPtr, "%d", &NumA[i] ) ;
                                for ( i=1; i<=Nmach; i++ ){
                                        for (j=1; j<=NumA[i]; j++){
                                                fscanf( cfPtr, "%d", &SNA[i][j] );
                                                fscanf( cfPtr, "%d", &ENA[i][j] );
                                        }
                                }
                        JobType = j ;
                }
        }
```

```
/*********Reading Data from the Files*********/
        if ( InputCondition == 1 ){
                if ( ( cfPtr = fopen ("Secondinput.txt", "r" ) ) == NULL)
                        printf( "File couldnot be opened\n");
                else {
                        fscanf(cfPtr, "%d", &Nmach);
                        fscanf(cfPtr,"%d", &JobType);
                        for ( i=1; i<=JobType; i++ )
                                fscanf(cfPtr,"%d",&y[0][i]);
                        for (i=1;i<=JobType; i++)
                                fscanf(cfPtr,"%d",&Process[i]) ;
                        for (i = 1; i<=Nmach; i++)
                                fscanf(cfPtr, "%d", &NumA[i] ) ;

                                for ( i=1; i<=Nmach; i++ ){
                                        for (j=1; j<=NumA[i]; j++){
                                                fscanf( cfPtr, "%d", &SNA[i][j] );
                                                fscanf( cfPtr, "%d", &ENA[i][j] );
                                        }
                                }
                }
        }
        if ( ( cfPtr2 = fopen( "Secondinput.txt", "w" ) ) == NULL )
                printf( "File couldnot be open\n" );
        else {
                fprintf(cfPtr2, "%d\n", Nmach );
                fprintf(cfPtr2, "%d\n", JobType );
                for (i=1; i<=JobType; i++)
                        fprintf(cfPtr2,"%d\n",y[0][i]);
                for (i=1; i<=JobType; i++)
                        fprintf(cfPtr2,"%d\n",Process[i]);
                for (i = 1; i<=Nmach; i++)
                        fprintf(cfPtr2, "%d\n", NumA[i] ) ;
                for ( i=1; i<=Nmach; i++ ){
                        for ( q=1; q<=NumA[i] ; q++ ){
                                fprintf( cfPtr2, "%d\n", SNA[i][q] );
                                fprintf( cfPtr2, "%d\n", ENA[i][q] );
                        }
                }
        }
                fclose (cfPtr2);
                c0=clock ();

/*******************************LPT2*******************************/

                for ( j = 1; j<=JobType; j++ )
                        v[j] = y[0][j];
                sum_v=0;
                for ( j=1; j<=JobType; j++ )
                        sum_v=sum_v+y[0][j] ;
                s=1;

C:
                LPT ();
                Flag_LPT = 1 ;
                y[m][s] = y[m][s] + 1;
                y[0][s] = y[0][s] - 1;
                LPT () ;
                if ( y[0][s] > 0 )
                        goto C;
                else
                        s=s+1;
                if (s<=JobType)
                        goto C;
```

```
                else{
                        for ( i=1; i<=Nmach; i++ ) {
                                Lmachine[i]=0;
                                for ( j=1; j<=JobType; j++ )
                                        Lmachine[i] = Lmachine[i] + Process[j] * y[i][j];

                                for (q=1; q<=NumA[i]; q++){
                                        if ( Flag_Av[i][q] == 1 )
                                                Lmachine[i] = Lmachine[i] + ENA[i][q] - SNA[i][q];
                                }
                        }
                }

/*******************************Exact Algorithm*********************************/
                Maximum () ;
                UB_LPT = max;
                UB = max;
                for ( j=1; j<=JobType; j++ )
                        y[0][j] = v[j] ;
                sum_v = 0;
                for ( j=1; j<=JobType; j++ )
                        sum_v = sum_v + (Process[j]*y[0][j]);
                LB= sum_v - ( Nmach-1 ) * UB;
                m=1;
                for (j=1; j<=JobType; j++)
                        sum_T = sum_T + Process[j]*y[0][j];
                UB_AV = UB ;
                Nmach_Back =  Nmach ;
                Lemma_2 () ;
                Flag_Th5 = 0 ;
                Sjob = 0 ;
                Flag_Th5_2 = 0 ;
                Machine_Th5_Back = 0 ;
B:
        if (m != Nmach)
                Lemma_1 () ;
        UB_2 = UB ;
        Sum_A=0;
        Sum_LB = 0 ;

        if (m==Nmach){
                for ( j=1; j<=JobType; j++ )
                        y[m][j] = y[0][j];

                for ( j=1; j<=JobType; j++ )
                        Sum_A = Sum_A + (y[m][j] * Process[j]);
                Sum_LB = Sum_A ;
                for ( q=1; q<=NumA[m]; q++ ){
                        if ( Sum_A > SNA[m][q] )
                                Lmachine[m] = Lmachine[m] + ENA[m][q] - SNA[m][q];
                }
        }

        else {
                if (  Machine_Th5_Back == m  )
                        ProcessSwap () ;


                for ( j=1; j<=JobType; j++){
                        k = UB_2 / Process[j];
                        if ( k<=y[0][j] ){
                                y[m][j] = k;
                                UB_2 = UB_2 - k*Process[j];
                                Sum_A = Sum_A + ( y[m][j] * Process[j] );
```

```
                                Sum_LB = Sum_A ;
                        }
                        else{
                                y[m][j]=y[0][j];
                                UB_2 = UB_2 - y[0][j] * Process[j];
                                Sum_A = Sum_A + ( y[m][j] * Process[j] );
                                Sum_LB = Sum_A ;
                        }
                }
        }

        if ( m!=Nmach )
                Restore () ;

        if ( UB != UB_AV )
                Sum_A = Sum_A + N[m] ;

        condition_1 = 0 ;

        if ( ( Sum_A > UB ) && ( m==Nmach ) ){
                for ( j=1; j<=JobType; j++ )
                        y[0][j] = y[m][j];
                m = m - 1 ;
                UB = UB_AV ;
                Lemma_1 () ;
                LoadingB ();
                condition_1 = 1;
                goto E;
        }

        if ( m != Nmach ){
                Theorem_1 ();
                if ( (TH_1 > UB_AV)  ){
                        Theorem_4 () ;
                        if ( TH_4 > UB_AV )
                                goto A ;
                        Theorem_2 ();
                        if ( TH_2 > UB_AV ){
                                if (m==1)
                                        goto A ;
                                else{
                                        Theorem_5_P3 () ;
                                        if ( m == 0)
                                                goto A;
                                        else{
                                                condition_1 = 1 ;
                                                goto E ;
                                        }
                                }
                        }
                        else{
                                Theorem_5 () ;                  //Theorem 5
                                y[0][JobType] = y[0][JobType] + y[m][JobType] ;
                                y[m][JobType] = 0 ;
                                LoadingB ();
                                condition_1=1;
                                goto E;
                        }
                }
        }

E:
                if ( condition_1 == 1 ){
```

```
                if  (Process[1]<Process[2]){     //Theorem 5
                        Theorem_5_P2 () ;
                        if ( m==0 )
                                goto A ;
                        else
                                if ( Flag_Th5_3 == 1 )
                                        goto E ;
                }

        if ( m ==0 )
                goto A;
        else
                if ( (m>1) && (Sum_B<=0) ) {
                        if (  Flag_Th5_2 == 1 ){
                                Flag_Th5_2 = 0 ;
                                Machine_Th5_Back = 0 ;
                                Flag_Th5 = 0 ;
                                Sjob_Back = 0 ;
                                m = m - 2 ;
                                if ( m==0 )
                                        goto A ;
                                Restore_P2 () ;
                                goto E ;
                        }

                        if ( Flag_Th5==1 ){
                                        Flag_Th5 = 0 ;
                                        Sjob_Back = Sjob ;
                                        Job_Th5_Back = Job_Th5 ;
                                        Machine_Th5_Back = Machine_Th5 ;
                        }

                        else{
                                Sjob_Back = 0 ;
                                Job_Th5_Back = 0 ;
                                Machine_Th5_Back = 0 ;
                        }

                        m = m - 1;
                        Restore_P2 () ;
                        goto E;
                }

                else
                        if ( (m == 1) && ( Sum_B<=0 ) )
                                goto A ;

        if ( m != Nmach )
                Theorem_1 ();
        if ( ( m != Nmach ) && ( TH_1 > UB_AV ) && (Sum_B > 0) ) {
                y[0][JobType] = y[0][JobType] + y[m][JobType] ;
                y[m][JobType] = 0 ;                                  //Theorem 3
                LoadingB ();
                goto E ;
        }
}
        if ( Flag_2 == 1 )
                goto A;
else
        if ( m == Nmach ){

                condition_1 = 0;

                for ( i=1; i<=Nmach; i++ )
```

```
        for ( j=1; j<=JobType; j++ )
                z[i][j]=y[i][j];

for ( i=1; i<=Nmach; i++ ){
        Lmachine[i] = 0;
        for ( j=1; j<=JobType; j++ )
                Lmachine[i] = Lmachine[i] + ( Process[j] * y[i][j] );
        if ( UB_AV != UB_Back[i] ){
                for ( q=1; q<=NumA[i]; q++ ){
                        if ( Lmachine[i] > SNA[i][q] )
                                Lmachine[i] = Lmachine[i] + ENA[i][q] -
                                SNA[i][q];
                }
        }
}

max=0;
for ( i=1; i<=Nmach; i++ ){
        if ( Lmachine[i] > max ) {
                max = Lmachine[i];
                m = i;
        }
}

        UB_Fsolution = max;
        UB = max - 1;
        UB_AV = UB ;
        LB = sum_T - (Nmach - 1)*UB ;
        if ( UB < LB )
                goto A;
        r = Nmach ; //part1

        for ( i=1; i<=Nmach; i++ ){
                if ( (SNA[i][1]==0) && (ENA[i][1]>= UB) ) {
                        if ( i != Nmach ){
                                for ( q=1; q<=NumA[i] ; q++ ){
                                        SNA[i][q] = SNA[Nmach][q] ;
                                        ENA[i][q] = ENA[Nmach][q] ;
                                }
                                Lmachine[i] = Lmachine[Nmach] ;
                                for ( j=1; j<=JobType; j++ )
                                        y[i][j] = y[Nmach][j] ;
                                for ( j=1; j<=JobType; j++ )
                                        z[Nmach][j] = 0 ;
                                Nmach = Nmach - 1 ;
                        }
                        else
                                Nmach = Nmach - 1 ;
                }
        }

        if (Nmach != r){
                max=0;
                for ( i=1; i<=Nmach; i++ ){
                        if ( Lmachine[i] > max ){
                                max = Lmachine[i];
                                m = i;
                        }
                }
        }

        Lemma_2 () ;

        if ( m == Nmach )
```

```
                                m = m - 1 ;
                        else
                                m = m ;

                        m = m - 1;
                        Restore () ;
                        m = m + 1 ;
                        Flag_Th5_2 = 0 ;
                        goto B;
                }
                else{
                        Restore ();
                        UB = UB_AV ;

                        if ( m==Machine_Th5_Back ){
                                Machine_Th5_Back = 0 ;
                                Job_Th5_Back = 0;
                                if ( Process[1] < Process[2] ){
                                        m = m + 1;
                                        ProcessSwap () ;
                                        m = m - 1 ;
                                }
                        }
                        if ( Machine_Th5_Back == (m+1) ){
                                for ( i=1; i<=JobType; i++ )
                                        Lmachine[m] = Lmachine[m] + Process[i]*y[m][i] ;
                                if ( Lmachine[m] > Load_Th5 ){
                                        Machine_Th5_Back = 0 ;
                                        Job_Th5_Back = 0 ;
                                }
                        }
                        Flag_Th5_2 = 0 ;
                        Flag_Th5 = 0 ;
                        m = m + 1 ;
                        goto B ;
                }

A:
                c1=clock();
                cputime=(c1-c0)/*(CLOCKS_PER_SEC)*/;

                if ( ( cfPtr = fopen( "..............txt", "w" ) ) == NULL )
                        printf( "File couldnot be open\n" );
                else {
                        fprintf(cfPtr,"\tElapsed CPU time test:   %f   millisec\n",cputime);
                        fprintf(cfPtr,"\n");
                        fprintf(cfPtr, "Initial Solution Makespan is = %d\n ", UB_LPT);
                        fprintf(cfPtr,"\n");
                        fprintf ( cfPtr, "Optimal Solution Makespan = %d\n", UB_Fsolution );
                        fprintf ( cfPtr,"Process Times              :");
                        for ( j=1; j<=JobType; j++ )
                                fprintf ( cfPtr, "%d ", Process[j] );
                                for ( m=1; m<=Nmach_Back; m++ )
                                        Print ();
                                fprintf(cfPtr,"\n");

                        sum_v = 0 ;
                        for (j=1; j<=JobType; j++)
                                sum_v = sum_v + v[j] ;
                        fprintf(cfPtr, "m = %d ;\n",Nmach);
                        fprintf(cfPtr, "N = %d ;\n",sum_v);
                        fprintf(cfPtr, "NA = %d ;\n", NumA[1] );
                        fprintf(cfPtr, "p=[");
                        for ( j=1; j<=JobType; j++ ){
```

```
                                while ( v[j] > 0 ){
                                        v[j] = v[j] - 1 ;
                                        fprintf(cfPtr, "%d ", Process[j] );
                                }
                        }
                        fprintf(cfPtr, "];");
                        fprintf(cfPtr,"\n");
                        fprintf(cfPtr, "s=[");
                        for ( i=1; i<=Nmach; i++ ){
                                fprintf(cfPtr,"[") ;
                                for ( q=1; q<=NumA[i] ; q++ )
                                        fprintf(cfPtr, "%d ", SNA[i][q] );
                                fprintf(cfPtr, "]");
                        }
                        fprintf(cfPtr, "];");
                        fprintf(cfPtr,"\n");
                        fprintf(cfPtr, "e=[");
                        for ( i=1; i<=Nmach; i++ ){
                                fprintf(cfPtr,"[") ;
                                for ( q=1; q<=NumA[i] ; q++ )
                                        fprintf(cfPtr, "%d ", ENA[i][q] );
                                fprintf(cfPtr, "]");
                        }
                        fprintf(cfPtr, "];");
                        fprintf(cfPtr,"\n");
                        j=1 ;
                        for (i=1; i<=m; i++){
                                if (NumA[i] > j)
                                        j = NumA[i] ;
                        }
                        fprintf(cfPtr, "M = %d ;\n", UB_Fsolution );
                }
                fclose (cfPtr);
                system("pause");
                return 0;

}
/*************************************LPT*****************************************/
void LPT (void)
{
        for (i=1; i<=Nmach; i++){
                Lmachine[i]=0;
                for ( j=1; j<=JobType; j++ )
                        Lmachine[i]= Lmachine[i] + ( Process[j] * y[i][j] );
                for ( q =1 ; q<=NumA[i] ; q++ ){
                        if ( Flag_Av[i][q] == 1 )
                                Lmachine[i] = Lmachine[i] + ENA[i][q] - SNA[i][q] ;
                }
        }
        Flag_q = 1 ;
        q = 1 ;
        while ( (q<=NumA[m]) && ( Flag_q == 1) ){
                if ( Flag_Av[m][q] == 0 ){
                        Flag_q = 0 ;
                        InitialAvailability () ;
                }
                q = q+1 ;
        }
        Flag_LPT2 = 1 ;
        MinimumLoad () ;
}
/*************************************Start Maximum*************************************/
void Maximum ( void )
{
```

```
        max=0;
        for (i=1; i<=Nmach; i++)
                if (Lmachine[i]>max)
        max=Lmachine[i];
}
/*************************************LoadingB*****************************************/
void LoadingB ( void )
{
        Flag_1 = 0;
        for ( j=JobType; j>=1; j-- ){
                if ( (y[m][j] > 0) && ( Flag_1==0 ) ){
                        y[m][j] = y[m][j] - 1 ;
                        for ( s = j - 1 ; s>=1; s-- )
                                y[m][s] = y[m][s];
                        UB_2 = UB;
                        for ( s=1; s<=j; s++ )
                                UB_2 = UB_2 - Process[s]*y[m][s];
                        for ( s = j+1; s<=JobType; s++ ) {
                                k = UB_2 / Process[s];
                                if ( k<=y[0][s] ) {
                                        y[m][s] = k;
                                        UB_2 = UB_2 - k*Process[s];
                                }
                                else {
                                        y[m][s] = y[0][s];
                                        UB_2 = UB_2 - y[0][s]*Process[s];
                                }
                        }
                        Flag_1 = 1;
                }
        }
        Sum_B = 0 ;
        for ( j=1; j<=JobType; j++ )
                Sum_B = Sum_B + y[m][j] * Process[j];
        Sum_LB = Sum_B ;
        Restore ();
}
/*************************************Restore*****************************************/
void Restore ( void )
{
        for (j=1; j<=JobType; j++)
                y[0][j] = v[j];
        for ( i=1; i<=m; i++ )
                for( j=1; j<=JobType; j++ )
                        y[0][j] = y[0][j] - y[i][j];
}
/*************************************Theorem_1*****************************************/
void Theorem_1 ( void )
{
        TH_1 = 0;
        for (i=1; i<=m; i++) {
                Lmachine[i] = 0 ;
        for ( j=1; j<=JobType; j++ )
                Lmachine[i] = Lmachine[i] + Process[j]*y[i][j];
        }

        sum_S=0;
        for ( i=1; i<=m; i++ )
                sum_S = sum_S + Lmachine[i];

        Sum_N = 0 ;
        for ( i= m + 1 ; i <= Nmach ; i++ )
                Sum_N = Sum_N + N[i] ;
        TH_1 = (float) ( sum_T - sum_S + Sum_N) / ( Nmach - m );
```

```
}

/************************************Theorem_2*****************************************/
void Theorem_2 ( void )
{
        TH_2 = 0;
        sum_S = sum_S - Lmachine[m] + UB_Back[m];
        TH_2 = (float) ( sum_T - sum_S + Sum_N) / ( Nmach - m );
}
/************************************Print*****************************************/
void Print (void)
{
        fprintf (cfPtr, "\n   ");
        fprintf (cfPtr, "\n   ");
        fprintf ( cfPtr, "Load Allocation for Machine [%d] ", m );
        for ( j=1; j<=JobType; j++ )
                fprintf (cfPtr, " %d   ", z[m][j]);
        fprintf (cfPtr, "\n   ");
        fprintf (cfPtr, "\n   ");
}

/************************************Start LPT2*****************************************/
void LPT2( void )
{
        for ( i=1; i<=Nmach; i++ ){
                y[i][s] = y[i][s] + 1 ;
                Lmachine[i] = 0 ;
                for ( j=1; j<=JobType; j++ )
                        Lmachine[i]= Lmachine[i] + ( Process[j] * y[i][j] );
                y[i][s] = y[i][s] - 1 ;

                for (q=1; q<=NumA[i]; q++){
                        if ( Flag_Av[i][q] == 0 ){
                                if ( (Lmachine[i] > SNA[i][q]) )
                                        Lmachine[i] = Lmachine[i] + ENA[i][q] - SNA[i][q];
                        }
                        else
                                Lmachine[i] = Lmachine[i] + ENA[i][q] - SNA[i][q];
                }
        }

        min = minimum;
        for ( i=1; i<=Nmach; i++ ){
                if ( Lmachine[i] < min ){
                        min=Lmachine[i];
                        m = i;
                }
        }
        y[m][s] = y[m][s] + 1 ;
}
/************************************IntialAvailability*****************************************/
void InitialAvailability (void)
{
        if (Flag_LPT ==1 ){
                if (Flag_LPT2 == 1){
                        y[m][s] = y[m][s] - 1 ;
                        Flag_LPT2 = 0 ;
                        LPT2 () ;
                        LPT () ;
                }
        }
        for (q=1; q<=NumA[m]; q++){
                if ( (Lmachine[m] > SNA[m][q]) ){
                        Flag_Av[m][q] = 1 ;
```

```
                        Lmachine[m] = Lmachine[m] + ENA[m][q] - SNA[m][q];
                }
        }
}

/*************************************Minimum*********************************************/
void MinimumLoad (void)
{
        min = minimum;
        for ( i=1; i<=Nmach; i++ )
                if ( Lmachine[i] < min ){
                        min=Lmachine[i];
                        m=i;
                }
}
/*************************************Lemma 1*********************************************/
void Lemma_1 (void)
{
        qStar = 0 ;
        for (q=1; q<=NumA[m]; q++){
                if ( (SNA[m][q]<=UB) )
                        qStar = q ;
        }

        if ( qStar > 0 ){

                if ( UB>=ENA[m][qStar] )
                        UB = UB - ENA[m][qStar] ;
                else
                        UB = UB - UB ;

                if ( (qStar-1) != 0  ){
                        for ( q=1 ; q<=qStar-1; q++ )
                                UB = UB - ENA[m][q] ;
                }
                for ( q=1; q<=qStar; q++ )
                        UB = UB + SNA[m][q] ;
        }

        UB_Back[m] = UB ;
}
/*************************************Lemma 2*********************************************/
void Lemma_2 (void)
{
        for ( i = 1 ; i<=Nmach ; i++)
                N[i] = 0 ;
        qStar = 0 ;
        for (i=1; i<=Nmach; i++){

                if ( UB_AV<=SNA[i][1]  )
                        N[i] = 0 ;
                else{
                        for ( q=1; q<=NumA[i]; q++){
                                if ( UB_AV > ENA[i][q] )
                                        qStar = q ;
                        }

                        if ( qStar == 0 )
                                N[i] = UB_AV - SNA[i][1] ;
                        else
                                if ( qStar == NumA[i] ){
                                        for ( q=1 ; q<=NumA[i] ; q++ )
                                                N[i] = N[i] + ENA[i][q] - SNA[i][q] ;
                                }
```

```
                                else
                                        if ( UB_AV < SNA[i][qStar+1] ){
                                                for ( q=1; q<=qStar ; q++ )
                                                        N[i] = N[i] + ENA[i][q] - SNA[i][q] ;
                                        }
                                        else{
                                                N[i] = UB_AV - SNA[i][qStar+1] ;
                                                for ( q=1; q<=qStar ; q++ )
                                                        N[i] = N[i] + ENA[i][q] - SNA[i][q] ;
                                        }
                        }
                }
        }
/************************************Theorem_4*****************************************/
void Theorem_4 (void)
{
        TH_4 = 0 ;
        Sum_UBback = 0 ;
        for ( i=1 ; i<=m ; i++ )
                Sum_UBback = Sum_UBback + UB_Back[i];
        TH_4 = (float) ( sum_T - Sum_UBback + Sum_N) / ( Nmach - m );
}
/************************************Theorem 5*****************************************/
void Theorem_5 (void)
{
        Flag_3 = 1 ;
        Flag_Th5_3 = 0 ;
        Flag_Th5 = 0 ;

        if ( Process[1] < Process[2] )
                Flag_Th5_3 = 1 ;

        if ( Flag_Th5_3 == 1 ){
                m = m + 1 ;
                ProcessSwap () ;
                m = m - 1 ;
        }

        m = m - 1 ;
        j = JobType + 1 ;
        while ( (Flag_3 == 1) && ( j != 1 ) ){
                 j = j - 1 ;
                if ( y[m][j]> 0){
                        Sjob = j ;
                        i = ( 1 + y[0][Sjob] + y[m+1][Sjob] )* Process[j];
                        if ( i <= UB_AV )
                                Flag_3 = 0 ;
                }
        }

        if (j==1){
                Lmachine[m] = 0 ;
                for ( i=1; i<=JobType; i++ )
                        Load_Th5 = Load_Th5 + Process[i]*y[m][i] ;
                if ( UB_Back[m]==Lmachine[m] )
                        Flag_Th5_2 = 1 ;
                m = m + 1 ;
        }
        else{
                m = m + 1 ;
                Job_Th5 = y[0][Sjob] + y[m][Sjob] + 1 ;
                Machine_Th5 = m ;
                Flag_Th5 = 1 ;
        }
```

```
        if ( Flag_Th5_3 == 1 ){
                m = m + 1 ;
                ProcessSwap () ;
                m = m - 1 ;
        }
}
/************************************ProcessSwap*****************************************/
void ProcessSwap (void)
{
        i = Process[1] ;
        Process[1] = Process[Sjob_Back] ;
        Process[Sjob_Back] = i ;
        i = v[1] ;
        v[1] = v[Sjob_Back] ;
        v[Sjob_Back] = i ;
        for ( i=0; i<=m-1; i++){
                Q = y[i][1] ;
                y[i][1] = y[i][Sjob_Back];
                y[i][Sjob_Back] = Q ;
        }
}

/************************************Theorem 5_P2*****************************************/
void Theorem_5_P2 (void)
{
        Flag_Th5_3 = 0 ;
        if ( y[m][1] < Job_Th5_Back )
                Flag_Th5_3 = 1 ;

        if ( Flag_Th5_3 == 1 ){

                ProcessSwap () ;

                if ( Flag_Th5_2==1 ){
                        Flag_Th5_2 = 0 ;
                        Machine_Th5_Back = 0 ;
                        Sjob_Back = 0 ;
                        Flag_Th5 = 0 ;
                        m = m - 2 ;
                }
                else{
                        m = m - 1 ;
                        Sjob_Back = Sjob ;
                        Job_Th5_Back = Job_Th5 ;
                        Machine_Th5_Back = Machine_Th5 ;
                        Flag_Th5 = 0 ;
                }

                Restore_P2 () ;
        }
}
/************************************Restore_P2*****************************************/
void Restore_P2 (void)
{
        UB = UB_AV ;
        Lemma_1 () ;
        Restore () ;
        LoadingB ();
}
/************************************Theorem_5_P3 *****************************************/
void Theorem_5_P3 (void)
{
        Flag_Th5_3 = 0 ;
```

```
        Flag_3 = 1 ;
        Flag_Th5 = 0 ;

        if ( Process[1] < Process[2] )
                Flag_Th5_3 = 1 ;

        if (Flag_Th5_3 == 1 ){
                m = m + 1 ;
                ProcessSwap () ;
                m = m - 1 ;
        }

        m = m -  1 ;
        j = JobType + 1 ;

        while ( (Flag_3 == 1) && ( j != 1 ) ){
                 j = j - 1 ;
                if ( y[m][j]> 0){
                        Sjob_Back = j ;
                        i = ( 1 + y[0][Sjob_Back] + y[m+1][Sjob_Back] )* Process[j];
                        if ( i <= UB_AV )
                                Flag_3 = 0 ;
                }
        }

        if (j==1){
                Lmachine[m] = 0 ;
                for ( i=1; i<=JobType; i++ )
                        Lmachine[m] = Lmachine[m] + Process[i]*y[m][i] ;
                if ( UB_Back[m]==Lmachine[m] )
                        m = m - 1 ;
        }
        else{
                Job_Th5_Back = y[0][Sjob_Back] + y[m+1][Sjob_Back] + 1 ;
                Machine_Th5_Back = m + 1 ;
        }
        Restore_P2 () ;
}
```

Appendix B

ILOG Input Model

B.1 Model 1

```
/*********************************************
 * ILP I
 * Author: Navid Hashemian
 * Creation Date: Sep 8, 2009 at 2:14:44 PM
 *********************************************/

int N=...;
int m=...;

range Process = 1..N;
range Machine = 1..m;

int p[Process]=...;
int s[Machine]=...;
int e[Machine]=...;
int M = ...;

dvar boolean x[Machine][Process];
dvar boolean Y[Machine] ;
dvar int+ h;

minimize
h;

subject to{
        forall (i in Machine) ct1:
                sum ( j in Process ) p[j]*x[i][j]<= s[i]*Y[i]+M*(1-Y[i]) ;

        forall (i in Machine)
                ct2:
                        sum (j in Process)p[j]*x[i][j] + (e[i]-s[i])*(1-Y[i]) <= h;

        forall (j in Process)
                        ct3:
                                sum (i in Machine) x[i][j] == 1 ;
}
```

B.2 Model 2

```
/*********************************************
 * ILP II
 * Author: Navid Hashemian
 * Creation Date: Jan 7, 2010 at 2:14:44 PM
 *********************************************/

int N=...;
int m=...;
int NA = ... ;

range Process = 1..N;
range Machine = 1..m;
range Availability = 1..NA ;
range Availability_2 = 1..NA-1 ;

int p[Process]=...;
int s[Machine][Availability] =...;
int e[Machine][Availability] =...;
int M = ...;

dvar boolean x[Machine][Process];
dvar boolean y[Machine][Availability] ;
dvar int+ h;

minimize
h;

subject to{
        forall (i in Machine)
                ct1:
                        sum ( j in Process ) p[j]*x[i][j] +
                        sum(q in Availability_2)(sum(k in 1..q)(e[i][k]-s[i][k])*y[i][q])<=
                        sum(q in Availability)(s[i][q]*y[i][q])+
                        M*(1-(sum(q in Availability)y[i][q])) ;

        forall (i in Machine)
                ct2:
                        sum (j in Process)p[j]*x[i][j] +
                        sum(q in Availability_2)(sum(k in 1..q) (e[i][k]-s[i][k])*y[i][q+1])+
                        sum(q in Availability)(e[i][q]-s[i][q])*
                        (1-(sum(q in Availability)y[i][q]))<= h;

        forall (j in Process)
                ct3:
                        sum (i in Machine) x[i][j] == 1 ;

        forall (i in Machine)
                ct4:
                        sum(q in Availability)y[i][q] <= 1 ;
}
```

Appendix C

Mersenne Twister

```
/************************* MERSENNE.CPP ******************** AgF 2001-10-18 *
*  Random Number generator 'Mersenne Twister'                                *
*                                                                            *
*  This random number generator is described in the article by               *
*  M. Matsumoto & T. Nishimura, in:                                          *
*  ACM Transactions on Modeling and Computer Simulation,                     *
*  vol. 8, no. 1, 1998, pp. 3-30.                                            *
*                                                                            *
*  Experts consider this an excellent random number generator.               *
*                                                                            *
*****************************************************************************/

#include "randomc.h"


void TRandomMersenne::RandomInit(long int seed) {
  // re-seed generator
  unsigned long s = (unsigned long)seed;
  for (mti=0; mti<N; mti++) {
    s = s * 29943829 - 1;
    mt[mti] = s;}}


unsigned long TRandomMersenne::BRandom() {
  // generate 32 random bits
  unsigned long y;

  if (mti >= N) {
    // generate N words at one time
    const unsigned long LOWER_MASK = (1u << R) - 1; // lower R bits
    const unsigned long UPPER_MASK = -1 << R;       // upper 32-R bits
    int kk, km;
    for (kk=0, km=M; kk < N-1; kk++) {
      y = (mt[kk] & UPPER_MASK) | (mt[kk+1] & LOWER_MASK);
      mt[kk] = mt[km] ^ (y >> 1) ^ (-(signed long)(y & 1) & MATRIX_A);
      if (++km >= N) km = 0;}

    y = (mt[N-1] & UPPER_MASK) | (mt[0] & LOWER_MASK);
    mt[N-1] = mt[M-1] ^ (y >> 1) ^ (-(signed long)(y & 1) & MATRIX_A);
    mti = 0;}

  y = mt[mti++];

  // Tempering (May be omitted):
  y ^=  y >> TEMU;
  y ^= (y << TEMS) & TEMB;
  y ^= (y << TEMT) & TEMC;
  y ^=  y >> TEML;

  return y;}
```

```
  // output random float number in the interval 0 <= x < 1
  union {
    double f;
    unsigned long i[2];}
  convert;
  // get 32 random bits and convert to float
  unsigned long r = BRandom();
  convert.i[0] =  r << 20;
  convert.i[1] = (r >> 12) | 0x3FF00000;
  return convert.f - 1.0;}


long TRandomMersenne::IRandom(long min, long max) {
  // output random integer in the interval min <= x <= max
  long r;
  r = long((max - min + 1) * Random()) + min; // multiply interval with random and truncate
  if (r > max) r = max;
  if (max < min) return 0x80000000;
  return r;}
```