UNIVERSITY OF BAHRAIN

College of Information Technology



Scheduling of Identical Parallel Machines under Availability Constraint

A Thesis Submitted in Partial Fulfillment of the Requirements for the Master Degree in Information Technology

Submitted by:

Elham Mohammed Iskandarnia

University Number: 19927274

Supervised by:

Dr. Youssef Harrath

(Assistant Professor)

University of Bahrain

Kingdom of Bahrain

June, 2016

DECLARATION

I declare that this work is the result of my own investigation and that it has not already been accepted in substance for any degree, nor is it currently submitted for any degree.

Signed: ---------- Date: 3,6,2014

Elham Mohammed Iskandarnia (Candidate)

The Thesis defense committee considers the Thesis titled:

Scheduling of Identical Parallel Machines under Availability Constraints

Submitted by Elham Mohammed Iskandarnia to the College of Information Technology is satisfactory and acceptable for the Master of Science in Information Technology.

Thesis Defense Committee:

Dr. Youssef Harrath
(Supervisor)

Department of Computer Science
University of Bahrain

Prof. Sadiq M. Sait
(Member)

King Fahd University of Petroleum & Minerals (KFUPM), Saudi Arabia

---

Dr. Amjad Mahmood
(Member)

Department of Computer Science
University of Bahrain

Abstract

Resource planning, optimizing resource utilization and cost, is important in industrial operation management. The domain of scheduling is extremely broad, including many fields like manufacturing, planning, time tabling, project management, software engineering, transportation, CPU job scheduling, cloud computing, service scheduling etc.

Resources and machines are subject to availability. The objective of this study is to develop algorithms to schedule jobs on parallel machines with target the minimization of the completion time. The jobs are resumable and arbitrary preemtion is permitted. This problem is considered a hard combinatorial problem and approximating techniques will be used to solve it.

The mixed integer quadratic programming will be introduced to solve the small size problems optimally and a constructive greedy algorithm will be introduced to find good feasible solutions.

The problem is formulated mathematically and a theoretical lower bound is proved. In addition to that, an absolute performance guaranty provided by a list scheduling algorithm is used to validate the performance of a proposed heuristic. The efficiency of the heuristic will be examined by calculating the distance of the obtained solutions from the lower bound and the value obtained by the list algorithm. For more validation of the new heuristic, we used the modified Graham data set.

Dedication

To:

Adel, for his unconditional support

My daughters Alaa and Ayeh, for continuing to believe in me

Acknowledgment

I would like to thank Dr. Youssef Harrath, not only for his time and extreme patience, but for his intellectual contributions to my development as a researcher.

I would also like to thank all my master instructors, Dr. Yusif Al Bastaki, Dr. Ahmed Mashour, Dr. Ahmed Zaki, Dr. Nabil Huwaihi and college dean Dr. Hesham Al Ammal for their guiding and supporting me over the years. You have set an example of excellence as researchers, mentors, instructors, and role models.

I would especially like to thank my amazing family for the love, support, and constant encouragement I have gotten over the years. In particular, I would like to thank my parents, my husband, and my daughters. You are the salt of the earth, and I undoubtedly could not have done this without you.

Table of Contents

Subject	Page number
Defense Committee's Approval	i
Abstract	ii
Dedication	iii
Acknowledgment	iv
Table of Contents	v
List of Tables	viii
List of Figures	x
List of Symbols and Abbreviations	xi
Chapter One Problem and Its Background	
1.1 Introduction	2
1.2 Statement of the Problem	4
1.3 Research Questions	4
1.4 Research Goals	5
1.5 Significance of the Study	6
1.6 Scope and Delimitation of the Study	7
Chapter Two Literature Review	
2.1 Complexity of Scheduling Problems & Their Solutions	10
2.2 Parallel Machine Scheduling For Minimizing	13

Makespan	
2.3 Parallel Machine Scheduling with Availability Constraints for Makespan Minimization	16
2.4 Additional Constraints In Parallel Machine Scheduling	21
2.5 Summary	22
Chapter Three Problem Modeling, Approaches To Solve the Problem	
3.1 Problem Description	25
3.2 Problem Complexity, Bounds And Space Analysis	29
3.3 Approaches To Solve The Problem	32
3.3.1 A Proposed Mathematical Formulation	32
3.3.2 New Heuristic	34
3.3.3 Constraint Programming	38
3.4 Research Methodology and Experimental Methods	39
Chapter Four Implementation, Experimental Results and Discussion	
4.1 Implementation	44
4.1.1 Implementation Of MIQP	44
4.1.2 Implementation Of The Heuristic	45
4.1.3 Implementation Of Constraint Programming	53

4.2 Experimental Results and Discussions	54
4.2.1 Small Size Problems	54
4.2.2 Big Size Problems	61
4.2.3 Approximation Ratio	66
Chapter Five Conclusion and Future Work	
5.1 Conclusion	69
5.2 Future Work	70
References	71

List of Tables

Number	Title	Page
1	Summary of literature review	22
2	Notation used to formulate the problem	28
3	Time complexity of the program: Best case	37
4	Time complexity of the program: Worst case	37
5	Parameter used in modified Graham Data Set Experiment	41
6	Processing times of the jobs: Array Job	47
7	Jobs sorted based on LPT	48
8	Preemption results stored in array Job	50
9	Program output: Array Assign	51
10	The completion time of each machine	51
11	Result of MIQP for limited size problems	55
12	Result of CP for limited size problems	55
13	Result of heuristic for limited size problems when the machine are sorted	56
14	Result of heuristic for limited size problems when the machine are not sorted	56
15	Comparison between the MIQP and the	57

	heuristic	
16	Comparison between the optimal solutions obtained by MIQP and makespan of CP	59
17	Summary of the results and setup methods	61
18	Upper/lower bound results	62
19	Comparison between the heuristic results versus MGDS	65
20	Summary of the relative performance gurantee of the heuristic	67

List of Figures

Number	Title	Page
1	Part of mathematical formulation implemented in ILOG CPLEX	45
2	Part of Java program implementing the heuristic	47
3	Parallel machines with their unavailable periods	48
4	Availability times of machines	49
5	The array T after the first iteration of the algorithm	49
6	The array T after preempting the job J_5	50
7	Gant Diagram showing the scheduking of fourtheen jobs in three parallel identical machines under availability constraints	51
8	Screen shot of program execution	52
9	Program output saved in text file	52
10	Sample of constraint programing with CPLEX	54
11	Closeness of the of the solutions provided by the heuristic to the optimal solutions	58
12	Closeness of the of the solutions provided by the CP to the optimal solutions	60
13	Closeness of the heuristic results to the lower bound	64
14	Closeness of heuristic to optimal solution obtained from Modified Graham data set	66

List of Symbols and Abbreviations

CP	Constraint Programming
C_L	C_{max} of the list algorithm
lb	Lower bound
MIQP	Mixed Integer Quadratic Programming
ub	Upper Bound
CO	Combinatorial Optimization
P_m	Parallel Machines
pmtn	Arbitrary preemption
$\rho_{Heuristic}$	Error band of heuristic
LPT	Longest Processing Time
SPT	Shortest Processing Time
MGDS	Modified Graham Data Set
CPLEX	IBMP CPLEX Optimization Studio.
C^{H}_{max}	Completion time of heuristic
C^{*}_{max}	Optimal solution

Chapter 1: Problem and Its Background

1.1 Introduction

1.2 Statement of the Problem

1.3 Research Questions

1.4 Research Goals

1.5 Significance of the Study

1.6 Scope and Delimitation of the Study

Chapter 1: Problem and Its Background

1.1 Introduction

A classical subject for industrial operation management is the resource planning, optimizing resource utilization, which has direct influence on cost, as this issue is essential for any business (Chen Z. L., 2004). Business organizations implement scheduling techniques in order to achieve maximum productivity to be competitive.

The domain of scheduling is extremely broad, and it can be used in many fields like manufacturing, planning, time tabling, project management, software engineering, transportation, CPU job scheduling, cloud computing, service scheduling etc. For example, in grid computing, by using effective and efficient algorithms for scheduling in the heterogeneous environment, achievement of promising potentials of tremendous distributed resources is guaranteed (Rostami, Pilerood, & Mahdavi, 2015).

Proper planning for a set of tasks to take place before execution with proper scheduling objective can guarantee that requirements and deadlines will be met (Pinedo M. L., 2009).

Productivity is essentially related to how well resources are optimized, reduction of waste, and efficiency. This type of optimization can be very complex because the number of inputs of a scheduling problem is very large and numerous constraints must be satisfied. Resource scheduling is one of the methods that can be implemented to obtain high productivity. In general, resource scheduling problems consists of:

(a) Set of resources;

(b) Set of processes to be scheduled on resources;

(c) Set of constraints that must be coped with; and

(d) Objective function(s).

A large number of industries and applications that widely utilize scheduling and planning is the main reason for researchers to be attentive to scheduling (Baker, 2009). A good scheduling method guaranties the best utilization of resources which ensures economical productivity and develops the business with competitive advantages (Xu, Liu, & Zhao, 2010). The target of scheduling is to optimize the use of the resources by choosing suitable objective function(s). The objectives of scheduling are chosen based on the business intentions and operational domains.

Businesses that provide services usually deal with different types of machines. Depending on the nature of the operation procedures, a specific job may or may not be processed by a certain machine or it should be processed in different stages by different machines. This means that a job should be processed by only one machine or it could be processed in all machines in order to be completed.

Basic scheduling problem assumes that machines are continuously available throughout the planning horizon. Of course, this is not the case in all situations; machines can be unavailable during the planning horizon due to many reasons like set up time, unavailability of operators or raw materials, maintenance and breakdown. Another reason that makes machines unavailable is the presence of special tasks. A special task is the one that should be processed in specific time. For example, consider an operating system for mono or multi-processors where sub-programs with higher priority will interfere with current program execution.

The duration and starting time of unavailability periods and job processing time may be known at the beginning of the planning horizon (offline) or be unknown on the starting and will be known during the job scheduling (online).

1.2 Statement of the Problem

This study will consider the scheduling of parallel machines with availability constraints to minimize completion time of resumable jobs when the scheduling environment is offline and arbitrary preemption is allowed. The problem can be formulated as:

$$(p_m, rs \mid pmtn \mid C_{\max}) \tag{1.1}$$

Explanation of notation is provided in Chapter 3 Section 1.

1.3 Research Questions

The objective of this study is to answer the following question: How to minimize the completion time when processing n jobs at m machines. The environment of planning is defined as follows:

- The machines are identical and parallel, which means the processing times of jobs in all machines are the same, and the scheduling is done offline, which means the processing times of all jobs are known in advance. Also, the starting time and duration of unavailability of each machine is known.
- The scheduling will be carried on in an arbitrary preemptive environment, which means that, if a job assigned to a machine is not completed before the starting of its unavailability, the job can be resumed in any available machine, not necessarily on the same machine.
- The jobs are resumable, which means if a job is interrupted because of unavailability of a machine, it can continue from where it stopped. In other words, after interruption, the remaining processing time of the interrupted job will be its processing time minus the time it spends before being stopped.

- The total time a job spends in all machines is not greater than its processing time.
- If m jobs are assigned to a machine then the sum of total time that all jobs spend at that machine must be equal or less than the objective function.

1.4 Research Goals

The aim of this study is to explore and develop a model to schedule n jobs at m machines when machines are subjects to availability constraints. The starting and duration of availability periods are known before scheduling periods. A detailed literature review will be conducted to understand the problem in order to provide a good solution. The goals of this study are to:

- Provide a mathematical formulation for the problem. A new Mixed Integer Quadratic Programming (MIQP) formulation of the problem will be proposed. A lower bound of the C_{max} will be introduced and proved mathematically.
- Use the IBM ILOG CPLEX studio software tool to implement and test the MIQP. MIQP will provide optimal solutions to limited-size problems.
- For larger sizes, a heuristic will be designed and elaborated to solve different problem instances. It will be implemented with Java Programming language. Java was selected as implementation language because of its characteristics such as easy to use, robustness cross-platform capabilities and security features (IBM, 2016).
- Constraint programming, which provide feasible solutions to the studied problem will be also developed, in order to measure and compare the efficiency of the heuristic.

- The heuristic will be evaluated and tested with different data sets and techniques. In addition, its efficiency will be tested by benchmarking it against available data set.

1.5 Significance of the Study

In many manufacturing and service industries, managers must decide for planning and scheduling to obtain business goal. This form of decision making which is done on regular bases plays an important role in the success of the business and its winning of markets in many business fields (Han, Zhang, Lu, & Lin, 2015). Mathematical techniques used for these types of decision-making, allocate limited resources to activities in such a way that the company's objectives are optimized and goals are achieved. The resources may be machines in factories, gates in airports, workers at construction sites, programmers working in software engineering projects or processing unit in computing environments. Jobs also may be operations in manufacturing lines, planes landing in airports, stages in construction or computer programs to be executed. Each job may have a starting point, due date and priority level. The objective of this decision-making can be in many different forms such as minimizing the time to complete all activities, maximizing the number of jobs finished before due date and minimizing the penalty paid for delay. As explained, the scope of this study is very wide and includes many industries. For instance, health care industry, hospitals in particular, faces many different scheduling problems engaging nurses, surgeons, operation rooms and expensive equipment. The objective is usually to optimize the utilization of resources.

In the real world, resources are not always identical. Machines need regular maintenance, which can be scheduled in advance, but they are subject to break down, which may happen unexpectedly at any time.

This situation can be handled by online scheduling. Also, machines are unavailable during setup time. For example in textile industry, for each different fabric the planes must be changed for machines, which work on waving of fabric.

In literature, there are researches treating the scheduling and planning, with unavailability constraints (Blazewicz & Drozdowski, 2000; Schmidt, 2000; Wang & Cheng, 2015) but the resumable job and preemptive environment are discussed in a few papers.

The result of this study can be used in situations referred to as controllable availability and by controlling availability of the machines; the cost of production can decrease (Dong, 2013). Today, demands for energy are increasing for all major economics and the cost of energy is one major factor for the development of nations. Industries consume a lot of energy in different countries, but in some situations, there are machines in manufacturing chains which are idle due to insufficient number of jobs. Keeping a machine in ON situation consumes energy. In addition, it cannot simply be off because there will be a re-setup cost for each time that machine will set up again. Therefore, a difficult decision that the production managers face is whether to shut down the idle machines or not. Also, if a machine is determined to be shut down, then the managers have to decide when to shut it down and how long the shutdown period should be. In the literature, this situation is referred to as controllable availability.

1.6 Scope and Delimitation of the Study

This study will discuss parallel machines environment in which a job can be processed in any available machines but all machines are each subject to one unavailability period. All machines have the same speed for all jobs and the jobs are independent. Also, the

environment is offline, which means that processing time of all jobs and availability periods of machines are known before the scheduling starts. If a job is interrupted before it finishes it can be resumed at any available machine from where it was stopped. It will be proved that the search space of such problem is very big and the problem is an NP hard problem. So, the scope of this study is scheduling of arbitrary preemptive jobs in parallel machines, when machines are subjected to maximum one unavailability in the scheduling period.

Chapter 2: Literature Review

2.1 Complexity of Scheduling Problems & Their Solutions
2.2 Parallel Machine Scheduling For Minimizing Makespan
2.3 Parallel Machine Scheduling with Availability Constraints for Makespan Minimization
2.4 Additional Constraints In Parallel Machine Scheduling
2.5 Summary

Chapter 2: Literature Review

In this chapter, some of the works done in literature which are related to scheduling problem with unavailability periods for minimizing makespan have been reviewed and explained. A brief overview of basic scheduling concepts has been also provided.

In the first section, the complexity of scheduling problems as well as the different approaches for solving scheduling problems has been studied. In the second section, parallel machines scheduling with minimizing makespan has been explained. In the third section, the availability constraint for parallel machines scheduling has been explained and some papers that covered this topic have been reviewed where, some constraints are added to classical scheduling problem. In the fourth section, new trends of parallel machine scheduling have been named and explained. The last part of the chapter has a summary table.

2.1 Complexity of Scheduling Problems & Their Solutions

The Scheduling problem is a subset of decidable and combinatorial problems. In computing theory decision problems (decidable) are the problems with yes or no as an answer. Gry & Johnso (1979) describe the complexity of a decision problem as follow:

- P: A problem is considered to be P if it can be solved in a polynomial time with its input size that is O(n).
- NP: (non determenistic polynomial size) A problem is NP if given the answer, and it is possible to verify that the answer is correct in time polynomial within the size of the input.
- NP-Hardness: is a problem as hard as hardest problem in NP. That means a Problem T is NP-hard if any problem Y in NP can be

transfered in polynomial time to T. In other words, a polynomial reduction will transfer Y to T.

- NP-Complete: A problem is considered NPC when it is NPH and NP. Although any given solution to an NP-complete problem can be verified quickly (in polynomial time), there is no known efficient way to find a solution at the beginning.

On the other hand, scheduling problem is categorized as a combinatorial problem. Combinatorial is derived from mathematical word combinatorics, which is a branch of mathematics, concerned with the study of arrangement and selection of discrete objects. So, finding the optimized answer to scheduling problems can be categorized under combinatorial optimization (CO). In applied mathematics, CO is a concept which explains, finding the best object from finite set of objects in answer space (Sait & Sharif, 2000). The word best means the object that best satisfies the objective function.

Even though the set of objects is a finite set, but it is usually very big set. For example, in listing of n jobs, $N!$ ordering is possible so answer space has $N!$ objects. Finding the best answer from this big space if not impossible, is not feasible in terms of time and computer resources. In this regards scheduling problem may have two types of solutions:

- **Feasible Solution:** Finding an object in the search space that satisfies all the constrains.
- **Optimal solution:** It is a feasible solution that has the best value among other feasible solutions for the objective function.

There are two types of algorithms that can be used to find the solution for combinatorial optimization solution (Sait & Sharif, 2000).

A. Exact Algorithms: An exact algorithm tries to find the best answer to the given problem. The most well-known among this category are:

- Linear Programming: This approach tries to formulate a problem as linear minimizing problem subjects to set of linear constraints which represent the convex polytope. The vertices of the polytope are the feasible solutions. The no of constraints to be satisfied usually are very big. For example, for n jobs and m machines, $n \times m$ constraints must be satisfied.
- Dynamic Programming: This method is very much suitable for problems whose answers may be viewed as results of sequence of decisions. This method always checks the early sequence and prunes the sequence that cannot lead to optimal solution so the full enumeration is not needed.
- Branch and Bound: This method explores the state space search tree in either a depth first or breadth first manner. Bounding functions are used to prune sub trees that do not contain optimal solutions.

B. **Approximation Algorithms**: Many optimization problems are considered NP-Hard. It is not possible to apply enumerative techniques for such problems. Instead, the heuristic (approximation algorithm) will be used to solve the NP hard problems. The well-known approximation algorithms are:

- Constructive greedy method: It starts from seed input and the other input is selected from succeeding order and added to partial solution until a complete solution is obtained.
- Local Search.
- Simulating Annealing.
- Genetic Algorithms.
- Tabu Search.
- Simulated Evolution.

2.2 Parallel Machine Scheduling For Minimizing Makespan

There are different objective functions for scheduling problems. One of the most popular performance measurements is the makespan minimization. The makespan denoted by $C_{\max}$ is the maximum completion time, which is the time interval between the starting of the first job on the system and the finishing point of the last job. Obviously, when there is only one machine and jobs are independent (no open shop) the solution for minimizing $C_{\max}$ is trivial.

In the literature, classical parallel machine problems consider series of identical machines, which are in general all ready at time zero and will always be available during the whole scheduling horizon. These machines must process number of independent jobs with different processing times. With the assumption that the preemption is allowed in the classical processing problem where machines are always available, the minimum makespan is obtained by:

$$C^*_{\max} = \sum_{j=1}^{n} p_j / m \tag{2.1}$$

Where n is the number of jobs and p_j is the processing time of the job j and m is the number of machines. When all the processing times are integers, the expression (2.1) of the makespan is optimal if $C^*_{\max}$ is also an integer number (Hashemian, Claver, & Béla, 2014). Obviously, the total processing time is consistently allocated between the machines. If the value of $C^*_{\max}$ is not an integer, then it becomes a lower bound.

If the preemption is not allowed, the classical machine problem with makespan minimization becomes a non-deterministic polynomial-time hard (NP-hard) problem. Lenstra and Rinnooy (1977) showed that

the problem $P_2 \| C_{\max}$ is NP-hard and the problem $P_m \| C_{\max}$ is strongly NP-hard. Garey and Johnson (1979) defined NP-Hard as class of decision problems which have succinct certificates (certificates whose size is bounded by a polynomial function of the input size) that can be verified in polynomial time. According to them, a problem Z is NP-hard if all problems in that class are reducible to Z. Garey and Johnson (1978) showed that from practical aspect, all NP-hard problems are not equally hard. Leung and Anderson (2004) set that the machine scheduling problem with availability constraints is NP-hard.

If a problem can be solved with a polynomial algorithm, then it is considered as an easy problem or traceable. According to Sait (Sait & Sharif, 2000) an algorithm is said to be polynomial if its time complexity is $O(p(n))$ where n is the problem size and $p(n)$ is a polynomial function of n. On the other hand, algorithms whose time complexity cannot be bounded by polynomial functions are called exponential algorithms. Their time complexity is $O(c^n)$.

Solving a problem with more than two machines is very challenging. Approximation methods or exact algorithms are used to solve such problems. Due to high computation requirements, dynamic programming can be used for small-sized problems only (Bazaraa, Jarvis, & Sherali, 2010). Eric (1998) and Bo (1998) showed that obtaining tight lower bound is very difficult when using the branch and bound algorithm for unavailability constraints.

Heuristics can be used to find a local feasible solution. One of the well-known heuristics in literature is List Schedule (LS). According to this method, jobs are sorted according to a predefined order, which refers to it as priority rules. The solution of makespan in LS depends on how the jobs are ordered. Mathematically, it is approved that for n jobs $n!$

permutations should be checked in order to find an optimal solution. Graham (1969) approved that the performance guarantee when applying a LS to classical parallel machine problem is:

$$C_{LS} / C^* \leq 2 - 1/m \tag{2.2}$$

Baker (2009) defined the 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 sub optimality of the makespan produced by a given heuristic procedure.*

Kaabi & Harrath (2014) listed the commonly used priority rules and algorithms in their survey. According to their survey the most important priority rules are:

- Shortest Processing Time rule (SPT): Jobs are arranged in ascending order of their processing times. Jobs with small values of processing times are given high priority to be processed first.
- Preemptive Shortest Processing Time rule (PSPT): This rule is a variation of SPT. At any time, the job with the minimum remaining time is to be processed first.
- Weighted Shortest Processing Time rule (WSPT): For some scheduling problems, each job $j, 1 \leq j \leq n$ is described by its processing time p_{ij} on the machine $M_i, 1 \leq i \leq m$ and a weight w_j indicating certain priority. According to the WSPT rule, the jobs are arranged in increasing order of the ratios p_{ij} / w_j.
- Earliest Due Date rule (EDD): Jobs are processed according to the increasing order of their due dates.
- Longest Processing Time rule (LPT): Jobs are arranged in decreasing order of their processing times. Jobs with large values of processing times are given high priority for scheduling.

- Largest Number of Successors rule (LNS): In the precedence graph, jobs with largest number of successors are given highest priority.
- Longest Remaining Processing Time first rule (LRPT): This rule is to be applied when the job preemptions are allowed. Jobs having longest remaining processing times are scheduled first.
- Longest Remaining Processing Time - First-Fastest Machine rule (LRPT-FM): This rule is to be applied when the job preemptions are allowed. Jobs having longest remaining processing times are assigned to the fastest machine.
- Shortest Remaining Processing Time rule (SRPT): This rule is to be applied when job preemptions are allowed. Jobs having shortest remaining processing time are scheduled first.
- List Scheduling (LS): Given a sequence of jobs, assign them one by one according to the list. Each job is assigned to the machine where the job can be finished as early as possible.

2.3 Parallel Machine Scheduling with Availability Constraints for Makespan Minimization

The machines are sometimes unavailable because of machine breakdown, preventive maintenance, or arrival of special jobs. So, it is always possible that a job cannot be completely processed on a machine during its availability. The job processing, therefore will be interrupted. Unavailability periods can be stochastic like machine breakdown and random arrival of special jobs. In these cases unavailability periods are decision variables which occur randomly and an online algorithm is needed to manipulate them. But, in case of maintenance or arrival of

special jobs with predetermined arrival times and durations, the problems can be solved by deterministic algorithms.

In section 2.2, the broad topic of scheduling of parallel machines for minimizing $C_{\max}$ has been discussed. In this section the narrower domain will be considered, which is scheduling of parallel machines with availability constrains. The unavailability of machines can happen in different patterns. In this section, first previous research works that investigated unavailability constraints for only one machine have been reviewed, then unavailability period at time zero will be reviewed and finally unavailability constraint within the planning horizon will be discussed.

A single machine with unavailability periods can be formulated as:

$$(1|r-a|C_{\max}) \tag{2.3}$$

Obviously, any arbitrary sequence will yield to the optimal makespan, but when other factors are added, the same result may not be obtained (Pinedo, 2012).

Lee (1991) studied parallel machine scheduling with availability constraints where one of the machines is always available and each machine has at most one unavailability period. This problem is referred as:

$$(P_m|r-a_{m-1,1}|C_{\max}) \tag{2.4}$$

In this case, durations of unavailability periods are different, but they all start at time zero. He mentioned that this problem can be considered as a classical parallel machine scheduling with release times. He introduced two algorithms to solve this problem. In the first algorithm, he assumed that the unavailability period of the machines starts at time zero as the jobs that are already assigned to the machine from previous zone. The author applied a classical LPT rule to assign the

jobs to the machines. He proved that the makespan provided by this rule is less than or equal to:

$$((\frac{3}{2}-1/(2m))C^*_{\max} \tag{2.5}$$

In the second method, he applied a Modified Longest Processing Time Rule (MLPT) which is a modified version of his first algorithm. In this method, unavailability periods are considered as special jobs and sorted along with other jobs in non-increasing order of their durations. The jobs and the unavailability periods are scheduled based on LPT but considering that each machine can do only one special job which means each machine has only one unavailability period. So, if a machine is assigned more than one special job, extra special jobs are redirected to the machines with the smallest scheduling process and with no special job. Then, the smallest processing time of the machine is removed and replaced with a special job and this job will be attached in head of unassigned jobs. He proved that the makespan of this algorithm is bounded by $C_{LPT} \leq \frac{4}{3}C^*_{\max}$.

Later, Lee et al. (1997) showed that the first algorithm performs poorly when the start of the unavailability period is not at time zero or when there is an unavailability period on all machines (all machines are unavailable at the same time). They showed that in such cases relative error of this algorithm can be arbitrary large even for two machines. They also proved that the LPT bound is not valid when a machine is unavailable for a whole period. That is when the number of the active machines is smaller than the total number of machines.

Liao and Lee (1997) modified the MLPT algorithm and improved Lee's bound to $C_{LPT} \leq \frac{5}{4}C^*_{\max}$.

Kellere (1998) developed a dual approximation algorithm using a bin packing approach for the equation 2.5 and got the same bound.

The research works mentioned above discussed the parallel machine scheduling with unavailability at time zero, which is referred to as release time. This problem is easier than unavailability periods that are not at zero time. This is true because of possibility of having job preemption when there are unavailability periods in the middle of the scheduling horizon. This makes the problem more difficult (Chen B. , 1998). Lee (1996) introduced an algorithm for $P_2 \mid r-a_1,1 \mid C_{\max}$ called LPT2. This algorithm is very similar to LPT algorithm. In LPT2, jobs are sorted based on LPT but are assigned to machine such that the completion time of job is minimized. Under the assumption that one machine is always available, the algorithm yields a performance guarantee of $C_{LPT2} \leq C^*_{\max} \frac{(1-\frac{1}{m})}{2} + C^*_{\max}$. That means for two machines $C_{LPT2} \leq \frac{5}{4} C^*_{\max}$.

Liao et al. (2005) found optimal makespan for $P_2 \mid r-a_1,1 \mid C_{\max}$ by partitioning it into four sub-problems and each one was solved optimally by an algorithm. But, this algorithm works on very limited models. They were able to obtain an optimal solution only for up to 100 jobs and two machines. Their method is not suitable for bigger models. Later, Liao and Sheen (2008) studied the problem of machines with different capabilities and proposed a method based on binary search algorithm which either verifies the infeasibility of the problem or determines the optimal schedule.

Tan and Young (2002) found an optimal solution for online algorithm with non-resumable jobs. They found the optimal solution for the two identical machines where the unavailability periods are not overlapping. They showed that for this scenario, the competitive ratio of

LS is 3 and the optimal algorithm has a competitive ratio of 5/2. Competitive ratio is a method for analyzing the online algorithm; it measures the ratio between online algorithm performance and offline algorithm performance for the same problem. So, according to this definition the proposed algorithm by is more competitive.

WU and Lee (2007) investigated single machine with unavailabilty when the jobs processing times are function of their starting times (Deteriorating jobs). The authors used binary integer programming techniques to solve the problem. Also, they showed that Shortest Processing Time heuristic (SPT) can be used to schedule this kind of jobs.

Wang and Cheng (2015) studied a situation when a firm should consider and decide to send some of the jobs to subcontractors to increase the number of on time jobs. They considered the firm as a machine which is always available and the subcontractor as a machine which is available on some time interval. They extended the L-Y heuristic by modifying backward adjustment and they applied Look head strategy and achieved the worst case ratio bound of 4/3.

The makespan in the literature is discussed along with other objective functions. Carlyle et al. (2001) and Han et al. (2015) considered multi objectives scheduling problem, with minimizing the makespan as one of the objectives. Gao et al. (2009) proposed a new parallel genetic algorithm (PIGA) based on vector group encoding methods and immune methods to solve the scheduling task on textile manufacturing for minimizing makespan and total tardiness penalty, and called it MOSIP.

2.4 Additional Constraints In Parallel Machine Scheduling

In the last decade the new constraint and limitation is added to the classical parallel machine scheduling. In this section, some research works including additional constraints on machines or jobs have been reviewed.

- Online scheduling: Problems where job processing times and machine unavailability are not known in advance have recently attracted many researchers. For example, Kalantari and Akbari (2008) studied online scheduling of parallel machine problem to benefit from promising prospective of incredible distributed resources of grid computer. They proposed methods based on queuing theory and neural network to solve the online scheduling problem.
- Resource availability: It means resources are subject to availability and processes have life time. Xu et al. (2010) proposed a greedy algorithm to solve the business scheduling problem when resource availability is added.
- Machine eligibility: It was introduced by Liao and Sheen (2008) and developed later by Leung et al. (2013). Eligibility means a job j cannot be processed on just any one of the m machines. Instead, it can be run in specific subset of machines which is referred to as processing set of job j.
- Learning effect: Traditionally, jobs processing times are assumed to be constant from the first job to be processed until the last job. However, as more products are manufactured, businesses get more experience and knowledge. This means that firms can produce more products in less time (unit cost will decline). Eren and Guner

(2009) proposed a mathematical model to solve multi-objective scheduling problem with learning effect constraints and minimize the total tardiness and makespan.

- Job deterioration effect: In most cases, raw materials are kept ready to be used when a job is scheduled in a machine. The raw materials waiting in queue for processing weaken over time, which can lead to their loss or decline of their value, this phenomenon is known as "Job deterioration effect". Learning Effect and Job Deterioration Effect were discussed by (Rostami, Pilerood, & Mahdavi, 2015). The authors considered a fuzzy environment to reflect uncertainty and ambiguity of parameters effecting the scheduling.
- Controllable Machine Availability: Controllable Machine Availability means a period of unavailability (when machine is off) can be scheduled in order to optimize jobs and machines availability operations. Dong (2013) used a branch and bound algorithm based on column generation approach to solve medium size problems optimally.

2.5 Summary

In this section, all reviewed papers have been summarized in Table 1. This table includes the problem notations, the algorithms used to solve, the algorithms' performances and the references.

Table (1)

Summary of literature review

Problem	Algorithm	Algorithm Performance	Reference
$1\|r-a_i\|C_{max}$	LPT	Exact	(Lee C., 1996)
$1\|r-a_i\|C_{max}$ with deteriorating job	Binary Integer Programming	Exact	(Wu & Lee, 2003)

$1\|r-a_i\|C_{\max}$ with deteriorating job	SPT	Exact	(Wu & Lee, 2003)
$P_m\|r-a_i\|C_{\max}$ At least one machine is always available and all unavailability periods start at time zero.	MLPT	$C_{LPT} \le \frac{4}{3}C^*_{\max}$	(Lee C., 1996)
$P_m\|r-a_i\|C_{\max}$ At least one machine is always available and all unavailability periods start at time zero.	MILP	$C_{LPT2} \le \frac{5}{4}C^*_{\max}$	(Liao & Lee, 1997)
$P_m\|r-a_i\|C_{\max}$ At least one machine is always available and all unavailability periods start at time zero.	Dual Approximation	$C_{LPT} \le \frac{4}{3}C^*_{\max}$	(Kellerer, 1998)
$P_m\|r-a_i\|C_{\max}$ At least one machine is always available	LPT2	$C_{LPT2} \le C^*_{\max}\frac{(1-\frac{1}{m})}{2}+C^*_{\max}$	(Lee C., 1996)
$P_2\|r-a_i\|C_{\max}$ One machine is always available	Partitioning and solve sub problems with TMO-based algorithms	Exact	(Liao, Shyur, & Lin, 2005)
$P_m\|r-M_j-a_i\|C_{\max}$	Binary Search Algorithm	Exact	(Liao & Sheen, 2008)
$P_2\|r-a_iJ\|C_{\max}$	Muntz and Gofman Algorithm	Exact	(Blazewicz & Drozdowski, 2000)
$P_2\|r-a_iJ\|C_{\max}$ The tasks are from chains	Linear programming Approach	Exact	(Blazewicz & Drozdowski, 2000)
Online Scheduling	Mathematical formulation and Neural Network	Exact	(Kalantari & Akbari, 2008)
$Pm\|LE\|\sum C+\sum T$ Learning effect	Mathematical formulation	Exact	(Eren & Guner, 2009)
$Pm, NC_{win}\|r_J, M_j\|C_{\max}$	Branch and bound	Exact	(Lee, Leung, & Pinedo, 2013)
$Q_m\|LE, D\|C_{\max}, ET$	Multi Objective branch and bound	Exact	(Rostami, Pilerood, & Mahdavi, 2015)

Chapter 3: Problem Modeling, Approaches To Solve the Problem

3.1 Problem Description

3.2 Problem Complexity, Bounds And Space Analysis

3.3 Approaches To Solve The Problem

3.3.1 A Proposed Mathematical Formulation

3.3.2 New Heuristic

3.3.3 Constraint Programming

3.4 Research Methodology and Experimental Methods

3.5 Data Sets

Chapter 3: Problem Modeling, Approaches To Solve the Problem

3.1 Problem Description

In general, a scheduling problem can be formulated as $\alpha|\beta|\gamma$ (Pinedo, 2012).

The α represents the planning environment, each task may be performed by any machine or a machine is dedicated for specific job. The environment can be defined as follow:

- Parallel machines: Any job can be processed by any machine, and no dependency or sequence is specified for jobs. This environment is divided into three categories (Chen B. , 1998):
  - Identical parallel machines; All the parallel machines are identical in term of their speed.
  - Uniform parallel machines: The parallel machines have different speeds. Consider $S_1, S_2, ..., S_m$ be the speeds of the machines $1, 2, ..., m$ respectively, with the relation $s_1 < s_2 < ... < s_m$. The processing time of a job j is given by p_j and the time it spends in each machine i is: $t_{ij} = p_j / S_i$.
  - Unrelated parallel machines: The processing time t_{ij} for a job j in machine i is arbitrary and have no special characteristic.
- Flow shop: A job consists of different tasks and the number of tasks for each job is the same as the number of machines. There are m machines in series. Each job has to be processed on each one of the machines. All jobs have to follow the same route, i.e., they have to be processed first on machine 1, then on machine 2, and so on. After completion on one machine a job joins the queue at the

next machine. Usually, all queues are assumed to operate under the First In First Out (FIFO) discipline, that is, a job cannot "pass" another while waiting in a queue.

- **Job shop**: The jobs pass through all machines but with different orders.
- **Open shop**: Machines have different speeds and all jobs must be processed in all machines with the same order.

If a job can be interrupted before it terminates its processing in a given machine, the job is said to be preempted. For example, if the processing time of a job falls in the unavailability period, it will be interrupted. Consequently the job will be preempted and the scheduling is called preemptive scheduling. After the unavailability periods, preempted job can be completed based on the type of preemption. Under arbitrary preemption, job can be send to any machine, but in operational preemption the interrupted job should be processed on the same machine it started on. The duration of a job after unavailability period depends on resumability or non-resumability of the job. A non-resumable job will be restarted from the beginning, but resumable job will be continued from where it was interrupted.

The field α includes $\alpha = \alpha_1, \alpha_2, \alpha_3, \alpha_4$ where:

- $\alpha_1 \in \{\phi, P_m, Q_m, R_m, F_m, J_m\}$ which is representing {single machine, identical parallel machines, uniform parallel machines, unrelated parallel machine, flow shop, and job shop};
- α_2 denotes the number of machines.
- α_3 describes different patterns of machine unavailability and it is formulated as: (Saidy, 2008)

  $\alpha_3 \in \{\phi, NC, NCzz, NCinc, NCdec, NCinczz, NCdeczz, NCsc\}$ which represents :

{Constant, Zigzag, Increasing, Decreasing, Increasing zigzag, Decreasing zigzag, Staircase}.

- α_4 mentions whether interrupted jobs are resumable (*rs*), semi-resumable (*sr*), non-resumable (*nr*), or a mixture between them (*rs/nr*).

The scheduling environment may have many limitations. These limitations are referred to as constraint. Examples of limitations are: due dates, storing costs, penalties and process with different priorities. These considerations are stated in the field β, which includes $\beta = \beta_1, \beta_2, ..., \beta_5$ where:

- $\beta 1 \in \{\phi, t - pmtn, pmtn\}$ which is representing {non-preemption, preemption, arbitrary preemption}.
- $\beta_2 \in \{\phi, r_j\}$ where ϕ indicates that all jobs are ready at time zero and r_j denotes jobs with different release dates.
- $\beta_3 \in \{\phi, d_j\}$ indicates whether the jobs come with due dates or not.
- $\beta_4 \in \{\phi, q_j\}$ indicates whether the jobs have tails.
- $\beta_5 \in \{offline, online\}$ denotes that the scheduling problem is either offline or online (Kaabi & Harrath, 2014).

The field γ describes the objective of scheduling. The objective function must minimized or maximized and often contains a single entry, even though as mentioned in literatures multi objective scheduling problems are very interesting research subject.

The notations used in this field γ are (Bazaraa, Jarvis, & Sherali, 2010; Pinedo, 2012):

- **Makespan ($C_{\max}$):** is defined as $\max(c_1, c_2, ..., c_n)$, is equivalent to the completion time of the last job to leave the system. A minimum makespan usually implies a good utilization of the machine(s).

- **Maximum Lateness** (L_{max}): is defined as $\max(L_1, L_2, ..., L_n)$. It measures the worst violation of the due dates.
- **Total weighted completion time:** The sum of the completion times. In the literature usually referred to completion time as the flow time. The total weighted completion time is then referred to as the weighted flow time.
- **Total weighted tardiness** ($\sum_j w_j T_j$): This is a more general cost function than the total weighted completion time.
- **Weighted number of tardy jobs** ($\sum_j w_j U_j$): It is a measure that can be recorded easily.

Accordingly, the formulation of the problem we studied is as mentioned in the equation 1.1. n resumable jobs have to be scheduled in m identical parallel machines. Arbitrary preemption is allowed, and the objective function is to minimize the completion time C_{max}.

Additional symbols to be used further are summarized in the Table 2.

Table (2)

Notations used to formulate the problem

Notation	Explanation
i	Index of a machine
j	Index of a job
m	Number of machines
n	Number of jobs
p_j	Processing time of job j
s_i	Starting of unavailability period of machine i
e_i	End of unavailability period of machine i

3.2 Problem Complexity, Bounds and Space Analysis

Lemma 1: The problem that we are studying and described by the formula (1.1) is NP-Hard.

Proof: In this problem it is assumed that preemption can happen when a job is assigned to a machine and its processing time is not finished before the start of an unavailability of that machine. This preempted job will be added to the job queue with modified processing time and a precedence constraint (the modified job cannot start before the interruption time. Therefore, after maximum m preemptions the problem will be $p_m \| C_{\max}$ Garey and Johnson (1978) showed that the problem $p_2 \| C_{\max}$ is NP-Hard and the problem $p_m \| C_{\max}$ is strongly NP-Hard. Also, Lee C. (1996) showed that $p_m | t-pmtn | C_{\max}$ is NP-Complete but, when unavailability constraint is added to the problem, the problem becomes NP-Hard.

Lemma 2: If we suppose that the unavailability period of every machine falls within the planning horizon, then a lower bound is proposed as:

$$C^*_{\max} \geq lb = (\sum_{j=1}^{n} p_j + \sum_{i=1}^{m} (e_i - s_i)) / m \qquad (3.1)$$

Where $C^*_{\max}$ is the optimal makespan.

Proof: If we consider the unavailability periods like jobs to be executed by the machines, the perfect case is when every machine processes jobs during the average of the total processing times and the unavailability periods. Since the unavailability periods have different durations, and start at different times, the problem becomes non trivial. In addition, the scheduling environment is preemptive and arbitrarily resumable which creates precedence constraints between the portions of one job processed in different machines making the makespan greater than the lower bound.

For our problem, if unavailability of machine starts after the machine finished its load, the duration of unavailability will not affect the makespan. Therefore, the lower bound can be tightened as:

$$C^*_{\max} \geq (\sum_{j=1}^{n} p_j + \sum_{i=1}^{m} (e_i - s_i) \times y_i)) / m \quad (3.2)$$

Theorem 1: The list scheduling performance guarantee cannot exceed the optimal value of the makespan by more than the maximum processing time of the jobs. This is given by the following expression:

$$C_L - C^*_{\max} \leq \underset{j=1}{\overset{n}{Max}} p_j - 1 \quad (3.3)$$

Where C_L is the makespan given by the list scheduling and $C^*_{\max}$ is the optimal makespan.

Proof: To prove the inequality in equation (3.3) the following method can be used to produce a solution to the problem:

1. No preferable order between the jobs or the machines.
2. Keep assigning the jobs in a given machine until the end of a job is equal or exceeds lb. If a job is interrupted by unavailability period, continue processing its remaining time in the same machine.
3. When the end of a job is equal or exceeds *lb* (defined in the equation (3.1)), in the current machine, move to the next machine and repeat the same process as in 2.
4. Stop when there are any more jobs to process.

The method guarantees that every machine has at most one job terminates after the lower bound. The worst case is in the machine in which the job that terminates after *lb* is the one that has the highest processing time. According to the step 3 of the above list scheduling, no job will start at *lb*.

This proves that the makespan will exceed the value of lb by at most $\underset{j=1}{\overset{n}{Max}} p_j - 1$. As a result,

$$C_L \le (\sum_{j=1}^{n} p_j + \sum_{i=1}^{m} (e_i - s_i) \times y_i) / m + \underset{j=1}{\overset{n}{Max}} p_j - 1 \quad (3.4)$$

And

$$(\sum_{j=1}^{n} p_j + \sum_{i=1}^{m} (e_i - s_i) \times y_i) / m \le C^*_{\max} \quad (3.5)$$

Both (3.4) and (3.5) prove that:

$$C_L - C^*_{\max} \le \underset{j=1}{\overset{n}{Max}} p_j - 1$$

As a result, from the previous lemmas and theorem, the target is to propose a heuristic that produces makespan C_H in the following range:

$$lb \le C^H{}_{\max} \le lb + \underset{j=1}{\overset{n}{Max}} p_j - 1 \quad (3.6)$$

The nearest value of C_H to lb, indicates that the new heuristic is performing well.

Lemma 3: The average search space for the studied problem is given by the equation (3.7).

$$Average\ (Search\ Space) = (m + \frac{n}{2})(\frac{n}{m})! \quad (3.7)$$

Proof:

- Best case: In ideal situation is when the number of jobs that each machine will process is equal to n/m. In case when no preemption happened (best case), the search space is $m(n/m)!$.
- Worst case: This case happens when the preemption happens for every machine. In this case, m new special jobs will appear. The average search space will be $m(\frac{n+m}{m})!$.
- Average case: It is trivial to find the average case.

3.3 Approaches To Solve The Problem

To solve the problem, three different approaches were proposed:

A. Mixed Integer Quadratic Programming (MIQP).

B. New heuristic.

C. Constraint programming.

The details of these methods will be explained in the following sections. However, the implementation, validation, testing and analysis will be provided in the next chapter.

3.3.1 Mixed Integer Quadratic Programming

To formulate the problem mathematically, we used the notations summarized in the Table 2. The problem is modeled as a Mixed Integer Quadratic Program that contains three elements: variables, objective function and list of constraints.

Variables

The following variables were used:

$$y_i = \begin{cases} 1 \text{ if the load of the machine } i \text{ terminates before } s_i \\ 0 \text{ elsewhere} \end{cases}$$

$$u_{ij} = \begin{cases} 1 \text{ if the job } j \text{ is preempted in machine } i \\ 0 \text{ if not} \end{cases}$$

$$x_{ij} = \begin{cases} 1 \text{ if the job } j \text{ is assigned to the machine } i \\ 0 \text{ if not} \end{cases}$$

t_{ij} = processing time the job j spends in machine i

M : A very big integer number

Objective function

$$\text{Minimize } C_{\max} = \max_{i=1}^{m} C_i \tag{3.8}$$

$$\text{Where } C_i = \sum_{j=1}^{n} t_{ij} + (e_i - s_i)(1 - y_i) \tag{3.9}$$

The equation (3.9) represents the load of a machine i to which the unavailability period is added if it is falls within the horizon production plan.

Subject to the constraints

$$\forall j = 1,...,n \quad \sum_{i=1}^{m} t_{ij} = p_j \tag{3.10}$$

The equation (3.10) indicates that the total time every job j spends in all machines is equal to its processing time. The preemption is also indirectly reflected in this constraint.

$$\forall i = 1,...,m \quad \sum_{j=1}^{n} t_{ij} + (e_i - s_i)(1 - y_i) \leq C_{\max} \tag{3.11}$$

The equation (3.11) indicates that the total load of a machine i added to its unavailability period is less than or equal to the objective function. The unavailability is excluded from the C_i when it falls after the termination of the last job j on that machine. This constraint is reflected in the equation (3.12).

$$\forall i = 1,...,m \quad \sum_{j=1}^{n} t_{ij} \leq (1 - y_i)M + s_i y_i \tag{3.12}$$

$$\forall i = 1,...,m \quad \forall j = 1,...,n \quad t_{ij} = x_{ij}(u_{ij} p_j - (u_{ij} - 1)(s_i - \sum_{k=1}^{j-1} t_{ik})) \tag{3.13}$$

The equation (3.13) indicates that if a job j is processed at a machine i without preemption, then its processing time on that machine is equal to p_j. Therefore, the processing time of that job in any other machine is equal to zero. However, if a job j is to be preempted in a machine i, it is processed on that machine until it becomes unavailable. The remaining processing time of the interrupted job j can be processed in the other available machines as stated in the equation (3.14).

$$\forall i = 1,...,m \quad \forall j = 1,...,n \sum_{k=1}^{i-1} t_{kj} + \sum_{k=i+1}^{m} t_{kj} = p_j - t_{ij} \tag{3.14}$$

The total time a job j spends in all machines is equal to its processing p_j and if it is preempted it cannot spend more than its processing time in all machines.

$$\forall j = 1,...,n \sum_{i=1}^{m} t_{ij} \leq p_j - M \sum_{i=1}^{m} (u_{ij} - 1) \quad (3.15)$$

The mathematical formulation we have proposed can be implemented using many available tools such as CPLEX and LINGO. The solutions provided by these tools are always optimal. The main concern for this method is the limited size of the problem instances to be solved since the search space size is exponentially growing with increase of number of jobs and machines.

3.3.2 New Proposed Heuristic

A New heuristic is designed using the constructive greedy algorithm in order to obtain a feasible solution from the search space. The solution is constructed step by step by assigning at a time the proper job to the right machine so that the jobs are assigned as early as possible and the preemption is avoided as much as possible.

When developing the new heuristic, the following points were considered:

- Sorting the jobs according to their processing times might help in decreasing the C_{max}.
- Sorting the machines according to their unavailability periods is another factor to include.
- Minimize the number of interruptions during the scheduling process. So, if there is a job that can start and finish during the availability of a machine, it will be processed at that machine to avoid interrupting longer jobs when the unavailability period starts.

- Machines will be busy during the whole planning horizon except during the unavailability periods. Thus the machines will be perfectly used without time waste or energy lost.
- The constraints of the problem have to be respected. No job will be processed at an unavailable machine and an interrupted job can continue in any available machine but not necessarily immediately.

The heuristic

2. Order the jobs according to a selected priority rule such as LPT and SPT.
3. Order the machines in ascending order of their unavailability durations.
4. Define an array T[m] to record the available times of the machines and initialize its values as following: For (i = 0,…,m-1), if (s_i = 0), then T[i] = e_i, else T[i] = 0.
5. For every job j
   5.1 Find the available machine i where i is the index of the minimum value in the array T.
   5.2 If the availability duration of the machine i is longer or equal than p_j, assign the job j to the machine i and update the array T.
      5.2.1 Else, if there is a job j' that can be assigned to the machine i without interruption ($p_{j'} \leq s[i]-T[i]$), assign it and update the array T.
      5.2.2 Else, process partially the job j on machine i as follow: $p_j = p_j - (s_i - T[i])$ and $T[i] = e_i$. After that, sort the jobs according to the step 1.
6. Repeat 4.1 and 4.2 until all jobs are assigned.

The first step of the heuristic treats the jobs using LPT, SPT or no special order. In fact, SPT and LPT are proved to be very useful for parallel machine problem scheduling. The second step orders the machines according to an ascending order of their unavailability periods. The machine with the lowest unavailability time will normally process more jobs. The initialization of the array T in step 3 shows whether the machine is available at time zero or if the unavailability of a machine i starts at time zero, the machine becomes available at time e_i. In addition to that, the array T records the completion time of every machine. The step 4 tries to assign the jobs wherever they can fit without interruption. Based on the job ordering given by step 1, three options may occur for the first non-processed job. Option 1 is when it can be processed completely in the first available machine. Second option occurs when its processing time is longer than the availability of the machine. In this case, that job will be temporarily ignored and the first next job that can be processed without interruption will be treated. Last case is to preempt the current job.

Time Complexity of the heuristic

The heuristic was implemented using Java 7.1 on a computer having the following characteristics:

- Single Processor.
- 64 bits.
- Run any arithmetic or logical operation in 1 unit of time.

When the number of machines is m and the number of jobs is n, the time complexity of the program is summarized in Tables 3 and 4. Two cases have been included: Best case when no job is interrupted and worst case when there are job interruptions in all machines. Big O and Big Ω notations will be used for the worst and best cases.

Table (3)

Time complexity of the program: Best case

Instruction	Ω	O	Space Complexity
Sorting Job	n	n^2	O(1)
Sorting Machine	m	m^2	O(1)
Initializing, assigning, reading from file	1	1	O(1)
Find minimum of array Job	1	m	O(1)
For all n jobs assign the jobs to machines	$n \times C_1$	$n \times (m+n+C_1)$	O(1)
Complexity	$\Omega(n)$	$O(n^2)$	O(1)

Table (4)

Time complexity of the program: Best case

Instruction	Ω	O	Space Complexity
Sorting Jobs	n	n^2	O(1)
Sorting Machines	m	m^2	O(1)
Initializing, assigning, reading from file	1	1	O(1)
Find minimum of array Job	1	m	O(1)
Assign jobs to machines (loop)	$E1$	$E2$	O(1)
Complexity	$\Omega(n+m)^2$	$O(n+m)^3$	O(1)

Where $E1 = (n+m)(C_1 m + C_2 n + C_3)$ and $E2 = [(n+m) \times (m+n+C_1 + (n+m)^2]$

The time complexity has cubic upper bound and linear lower bound.

3.3.3 Constraint Programming

Constraint programming is a technique that was used first in the Artificial Intelligence (AI) community. In recent years, it has often been implemented in combination with Operations Research (OR) techniques in order to improve its effectiveness (Frühwirth & Abdennadher, 2013). Constraint programming only tries to find a good solution that is feasible and satisfies all the given constraints, but not necessarily an optimal solution. Constraint programming intends to satisfy the relation between the variables which are stated in form of constraints. There are some programming languages used for constraints programming like Prolog, CHIP etc. CPLEX studio has predefined variables for modeling the constraints which are related to release dates, due dates ...etc. (Constraint Programming, 2016).

To use constraint programming techniques the model is redesigned as set of if-else conditions for same decision variables.

The *if-else* constraints are as follow:

1. If an unavailability period of a machine starts after processing the last job on that machine, it will not be considered in the makespan calculation.
2. The completion time of a machine is extracted from adding the total load of the machine to its unavailability period if the constraint in 1 is not satisfied.
3. If a job j is not assigned to a machine i, it cannot be preempted in machine i.
4. The total time of a job j spent in all machines is equal to its processing time.
5. A job j is interrupted in a machine i if it is assigned to that machine and its processing time is not terminated during the availability of the machine. This means part of the job j has to be completed after

the machine unavailability on the same machine or immediately in any other available machine.

6. A job can be assigned to more than one machine if it is preempted in one machine.

3.4 Research Methodology and Experimental Methods

Three different approaches were introduced to solve the problem. MIQP provides the optimal solution. *IBMP CPLEX Optimization Studio* software was used to program the mathematical formulation. This software is limited by the size of instances to solve and specially a maximum of 300 variables can be treated (IBM ILOG OPL Language, 2010).

The new heuristic was implemented using *JAVA 7.1*. The results which will be presented and discussed in Chapter 4 show that the heuristic is providing very good feasible solutions for small and big size problems.

The heuristic starts with a list of unassigned jobs and always tries to select the most suitable job to be processed in the suitable machine. Therefore, the way the jobs or the machines orders may affect the results. In order to examine the effect of different ways of ordering jobs in improving the results, LPT, SPT, or any order will be investigated. In addition to that, if the machines with short unavailability durations process more jobs, the result will be improved. Based on that, six cases will be tested as summarized down:

For the machines:

A. The machines are sorted based on ascending order of unavailability durations.

B. The machines are not sorted.

And for the jobs:

1. Jobs are sorted according to LPT
2. Jobs are sorted according to SPT.
3. Jobs are not sorted

The result of these different initializing steps will be demonstrated in Chapter 4 and the tests will be denoted as A1, A2, A3, B1, B2, and B3.

The third approach to solve the problem which is constraint programming was implemented by programming the designed constraints in CP engine of *IBMP CPLEX Optimization Studio.*

In order to compare and justify the effectiveness of the three proposed methods, the CPU time was measured and recorded. In addition, in order to examine the efficiency of our proposed heuristic, the distance of the generated solutions from the optimal solutions and the performance guarantee will be checked. In order to do this, the problem will be divided in to two parts: big size and small size. For the small size instances, the three methods will be used and their results will be compared. Whereas, for the big sizes, in order to validate the efficiency of the new heuristic, which cannot be solved with CPLEX, will be tested using the lower bound and the performance guarantee given in equations (3.6).

An additional experiment was set up based on the famous example given by Graham (1979). He proved that the performance ratio of listing problem for $P_m \| C_{\max}$ is at most $\frac{4}{3} - \frac{1}{3m}$.

As the parallel machine scheduling with availability constraints is very near to Graham model, it is natural to test the new heuristic with that class of problems. This type of experiment is performed in some researches (Blazewicz & Drozdowski, 2000; Saidy, 2008; Hashemian, Claver, & Béla, 2014). The optimal value of appropriate problem is *3m*

when there is no availability constraint on machines. Hashemian, Claver, & Béla (2014) run the modified version of Graham's expriment with unavailability period. Table 5 summerises the modified Graham expriement.

Table (5)

Parametrs used in modified Graham Data Set Expriement

Number of Jobs	$2m+1$
Number of Machines	M
Processing time of jobs	$2m-1, 2m-1, 2m-2, ..., m+1, m+1, m, m, m$
Number of unavailability periods	m (each machine 1)
Duration of unavailability	15

The results of the Modified Graham Data Set (MGDS) will be benchmarked with the new proposed heuristic.

3.5 Data set

The data used for testing and comparing the small size problems is created randomly. Data used for modified Graham test is created as mentioned in Table 5. To generate random numbers two methods were used.

- The Java class *Random_Number* was developed to create the data. In this class, java method *(Thread Local Random Current)* which is offered in java7, was used to create random processing times for all jobs. This method was offered in java 7 in *java.util.concurrent* class. The facility of defining the seed for *Thread Local Random Current,* and its ability to create bounded random number were implemented to create controlled and meaningful random numbers (IBM, 2016). In order to monitor the data, the numbers of machines, as well as the numbers of jobs are entered manually. The

same methods were used to create the starting and ending times of unavailability of each machine. The class ensures that the starting time is always less than the ending time for each machine.

- *Mersenne Twister Random Number Generator* was used to generate random numbers. It is common random number generator and expert considers it an excellent generator (Matsumoto & Nishimura, 1998). In order to use reliable data for each instance size, three different random sets of data are generated and tested. One set is created by using java class *Thread Local Random Current* and two sets by using *Mersenne Twister Random Number Generator.*

Chapter 4: Implementation, Experimental Results and Discussion

4.1 Implementation

4.1.1 Implementation Of MIQP

4.1.2 Implementation Of The Heuristic

4.1.3 Implementation Of Constraint Programming

4.2 Experimental Results and Discussions

4.2.1 Small Size Problems

4.2.2 Big Size Problems

4.2.3 Approximation Ratio

Chapter 4: Implementation, Experimental Results and Discussion

In Chapter 3, three approaches were introduced to solve the problem of scheduling resumable jobs on identical parallel machines with arbitrary preemption, when the environment is offline. In this chapter, the implementation of the three approaches will be discussed. Then, the heuristic will be validated for small size problems with the MIQP performance. For big size problems that cannot be solved with MIQP, theoretical bounds will be calculated to decide about the heuristic's performance. The heuristic will be also tested using the MGDS.

4.1 Implementation

4.1.1 Implementation of MIQP

MIQP provides the optimal solution. MIQP was solved by one of the best commercial solvers available *IBMP CPLEX Optimization Studio.* CPLEX implements optimization using sifting algorithm, primal simplex, dual simplex and barrier algorithm (IBM ILOG OPL Language, 2010). For this purpose, the mathematical model was coded using ILOG language.

In order to prevent excessive computational time whenever the problem is not solved by CPLEX within the time limit of 0-14,400 seconds (0-4 hours), we stopped the execution. The time limit is set by considering the maximum time that the new heuristic takes to solve the same problem instance (4 hours is more than 15,000 times bigger than the required time for the heuristic). The Figure 1 shows the relation between the written code in ILOG and the proposed mathematical formulation.

```
subject to {
forall ( j in Process ) ct1 :
sum ( i in Machine) t[i][j]==p[j] ; (3.11)
forall(i in Machine ) ct2 :
sum ( j in Process )t[i][j] + ( e [ i ]-s [ i ])*(1-Y[ i ] ) <= h ; (3.12)
forall(i in Machine ) ct3 :
sum ( j in Process )t[i][j]<= (1-Y[i])*M+ s[i] ;
forall (i in Machine)
sum(j in Process)t[i][j]<=h;
forall(i in Machine,j in Process)
t[i][j]>=0 && t[i][j]<=p[j];;
forall(j in Process)
sum(i in Machine)t[i][j]<=p[j]+M*sum( i in Machine)(u[i][j]-1); (3.13)
forall(j in Process)
sum(i in Machine)x[i][j]<=m;
forall (a in Machine,k in Process){
t[a][k]== x[a][k]*(u[a][k]*p[k]-((u[a][k]-1)*(s[a]-sum(j in 1..k-1)t[a][j])));
(sum(i in 1..a-1 )t[i][k] +sum(i in a+1..m )t[i][k])==p[k]-t[a][k]; (3.14)
}
```

Figure (1)

Part of the mathematical formulation implemented in ILOG CPLEX

4.1.2 Implementation of the Heuristic

To explain the implementation of the heuristic algorithm in details, a numerical example will be presented and the output of the program will be shown.

Preliminaries:

- A two dimensional array Job $(n\times 2)$ which holds job indices and their processing times accordingly.
- A two dimensional array machine $(2\times m)$ which holds the start and end of the unavailability periods of machines.

Data Structures:

- Array Assign: with maximum size $2m\times n$: If the job j is assigned to the machine i, then the program will modify the row i and row $i+1$. At the end of the program, the row i-1 shows the jobs are assigned to the machine i.

- Array T: with size m, which will indicate for every machine, the time it is ready to process a job. At the end of the program, T holds the makespan of every machine i.

Procedures

The Program has four main procedures:

- Void Set (j, i): This procedure is called when the availability of the machine i (machine [0] [i] – T [i]) is bigger than or equals to p_j (Job [1] [j]). The program will replace the first zero column of row i-1 with Job [0] [j] and row i with Job [1] [j]. It will also replace T [i] with (T [i] + Job [1] [j]) and will set Job [1] [j] =0.
- FillMachine: This procedure is called when the availability of the machine i (machine [0] [i] – T [i]) is less than p_j (Job [1] [j]). The procedure will look at the array Job and finds the job j' having a processing time $p_{j'}$ less than or equal to the availability of machine i then it will call Set for job j' and machine i.
- *FitJob*: It will check that if there is another available machine i' which job j can be set on it, then it will call *Set* (j, i').
- Preem_it: This procedure is called when the availability of machine i is less than p_j and there is no job to fit in machine i. The algorithm will replace the first non-zero column of row *i-1* of the array Assign with Job [0] [j] and row i with T[i]-machine [0] [i]. It will also replace T[i] with (T [i] + machine [1] [i]) and also will replace Job [1] [j] by p_j – the availability period of machine j.

Figure 2 contains a screen shot of the Java program showing the call of the above four procedures.

```
while(j<jr){

int min=  getMinValue(T);
int i= getMinindex(T,  min);
 H=machine_array[i][0]-min;
   if((Job[i][j]==H || H>Job[i][j] ||H<0) && (Job[i][j]!=0))
   {Set_it(j,i) ;}
   else if(H<Job[i][j] &&(Job[i][j]!=0)){
   logic1=FitJob(j,i);
   logic2=FillMachine(j,i,H);
   //System.out.println(logic1+" " +logic2);
   if((logic1==0) &&(logic2==0))
   {
        Premm_it(j,i);
Sort_array(Job);
                j=-1;
    }
   else if((logic1==0) &&(logic2==1))

   {            j=j-1;
   }
  \
```

Figure (2)

Part of the Java program implementing the heuristic

The following example will show how the algorithm is working

Numerical Example:

In this section, an example will be explained to demonstrate the implementation of the algorithm. The preliminaries of this example are: Assuming that 14 jobs will be scheduled on 3 machines, where the processing times of the jobs are generated randomly and shown in Table 6.

Table (6)

Processing times of the jobs: Array Job

J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9	J_{10}	J_{11}	J_{12}	J_{13}	J_{14}
10	20	10	20	20	20	8	8	7	5	5	5	1	5

The starting and ending of the unavailability periods of the three machines are listed below:

M_1= {12, 24}

M_2= {0, 10}

M_3= {20, 38}

First, the jobs will be sorted based on LPT rule like shown in Table 7.

Table (7)

Jobs sorted based on LPT

J_2	J_4	J_5	J_6	J_1	J_3	J_7	J_8	J_9	J_{10}	J_{11}	J_{12}	J_{14}	J_{13}
20	20	20	20	10	10	8	8	7	7	5	5	5	1

The machines will be renamed based on the durations of the unavailability periods as shown in Figure 3.

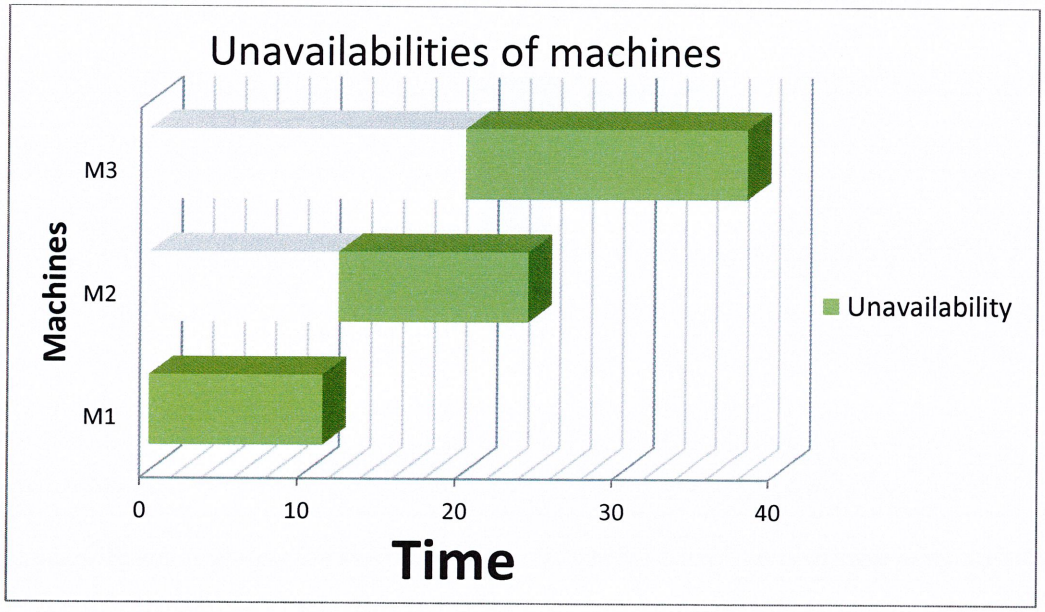


Figure (3)

Parallel machines with their unavailable periods

The array T will be initialized by values representing the times when every machine is available (Figure 4).

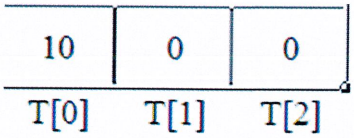


Figure (4)

Availability times of machines

At time zero, M_1 is unavailable whereas M_2 and M_3 are available. Since the first job duration is more than the availability period of M_2, the algorithm will try to find the job that set on M_2. So first, the job with duration 10 will be assigned to M_2, and since machine M_3 is available at the same time, the procedure *FitJob* will be called and J_2 will be assigned to M_3. Since the load of M_3 will end exactly at the beginning of unavailability period of M_3, the unavailability duration of M_3 will be added to T_3 and array T will be as shown in Figure 5.

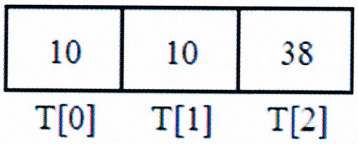


Figure (5)

The array T after the first iteration of the algorithm

The next job in the list is *J4*. It will be assigned to first available machine (M_1,) and the procedure *Set (2, 0)* will be called.

In next iteration of the algorithm, machine 2 (minimum value of T) that is available at time 10 will be selected. But, the first job in the queue (*J5*) cannot fit in M_2 because the availability of M_2 (12-10) is less than the processing time of J_5.

The algorithm will call *FillMachine* to search for the job witch a processing time less than or equal to the availability period of M_2. J_{13} is selected and *Set (13, 1)* is called. The program will trigger again at time 11, since there is no job that fits. The next job in the queue will be preempted and, the Job array will be sorted again as shown in Table 8.

Table (8)

Preemption results stored in array Job

J_6	J_5	J_3	J_7	J_8	J_9	J_{10}	J_{11}	J_{12}	J_{14}	J_2	J_4	J_1	J_{13}
20	19	10	8	8	7	7	5	5	5	0	0	0	0

After calling the procedure *Preem_it*, the array T will be modified as shown in Figure 6.

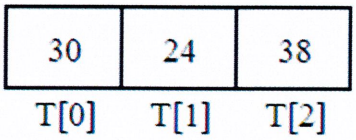


Figure (6)

The array T after preempting the job J_5

The program will trigger again at time 24. Since all the unavailability periods are previously considered, the jobs will be assigned via the *Set* procedure. At the end of the program, the final state of the array Assign is shown in Table 9.

Table (9)

Program output: Array Assign

Load M1	20	19	7	5	0	0	0	Duration
	4	5	9	11	0	0	0	Job index
Load M2	10	1	1	20	8	7	5	Duration
	3	13	5	6	7	10	14	Job index
Load M3	20	10	8	5	0	0	0	Duration
	2	1	8	12	0	0	0	Job index

The array T containing the makespan of each machine is given in the Table 10.

Table (10)

The completion time of each machine

59	64	61

The result of job scheduling of the 14 jobs on 3 machines provided by the heuristic is illustrated in FIgure 7. The result shows that only J_5 is interrupted in M2 and continues its processing in M1. The makespan $C_{\max} = 64$. Whereas, the optimal solution obtained by CPLEX is 62.

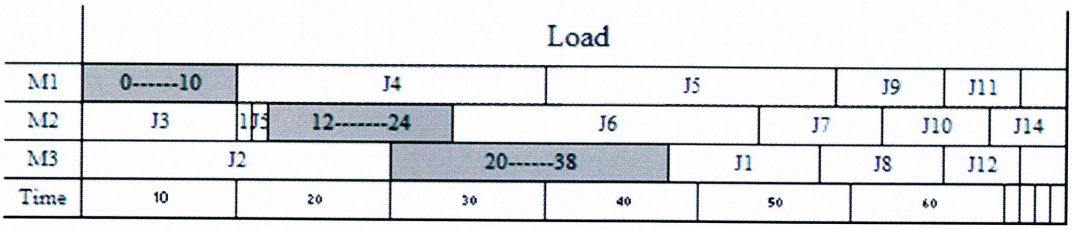


Figure (7)

Gant Diagram showing the scheduling of 14 jobs in 3 parallel identical machines under availability constraints

The algorithm is coded using *Java* (*NetBeans IDE 8.0.2*) by using a personal laptop with Inter Core i5 processor and Windows 2010 as operating system. The Figure 8 is showing two screens; part of the java code and the output screen.

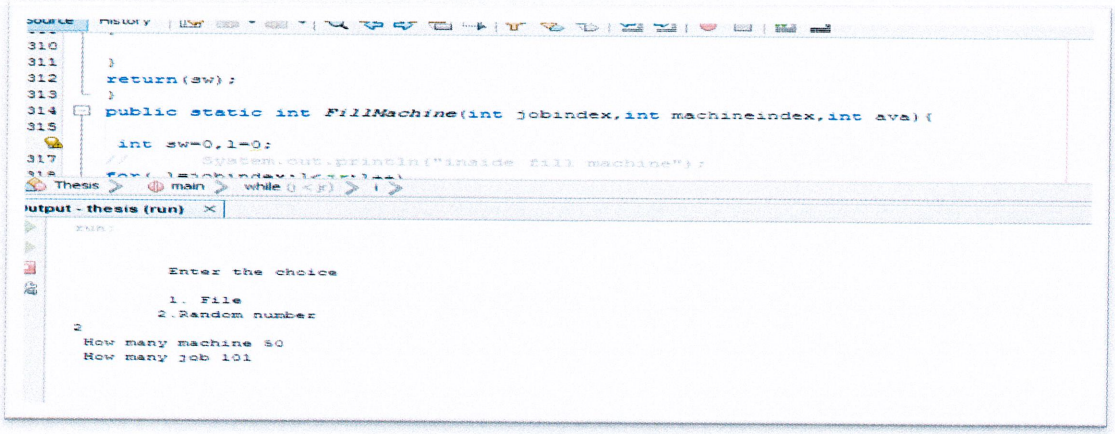


Figure (8)

Screen shot of program execution

The result of the program is recorded in a file showing the jobs assigned to each machine and the completion times. Figure 9 shows the output for the example explained before.

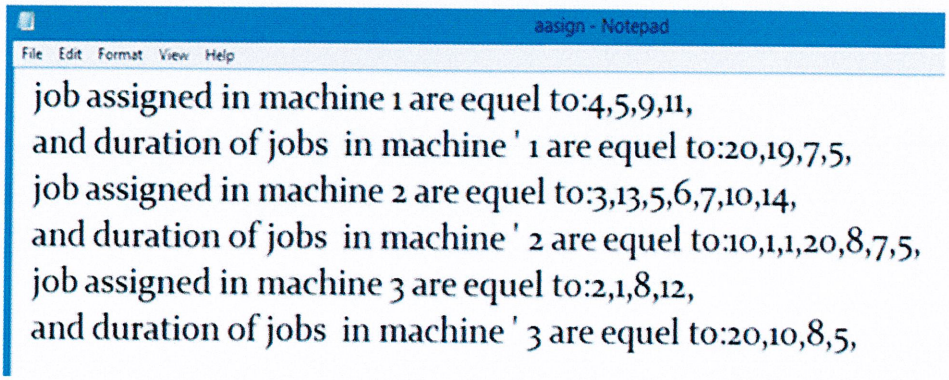


Figure (9)

Program output saved in a text file

4.1.3 Implementation of Constraint Programming

In order to check the performance of the heuristic, Constraint Programming (CP) was used. CP focuses on constraints and variables rather than objective functions. The constraints in general state the relation between unknown variables (decision variables) and logics to set the domain for each of them. Frühwirth & Abdennadher (2013) stated the advantages of constraint programming as:

- Declarative problem modeling on a solid mathematical basic.
- Propagation of decision effect of using efficient algorithm.
- Search for feasible solutions in efficient time.

The if-else constraints included in Chapter 3 were coded by CP engine of *IBMP CPLEX Optimization Studio.* The advantages of using CPLEX are (Constraint Programming, 2016):

- Aggressive elimination of possibilities.
- Rapidly traversing the decision tree.
- Adaptive Search.
- Awareness of time.

The figure 10 shows part of the code of the CP model.

```
constraints {
     forall (i in Machine) ct2:
  sum(j in Process)t[i][j]+ (e[i]-s[i])*(1-Y[i])<=h;
       forall(j in Process)
    sum(i in Machine)t[i][j]==p[j];
        forall(j in Process){
  sum (i in Machine)x[i][j]>=1;
  sum (i in Machine)x[i][j]<m;}
  forall(i in Machine,j in Process){
 (x[i][j]==0)=> u[i][j]==0;
  (x[i][j]==0) =>  t[i][j]==0;}
  forall(i in Machine,j in Process)
     (x[i][j]==1 && u[i][j]==0)=> t[i][j]==p[j];
    forall(i in Machine,j in Process)
     (u[i][j]==1)=> t[i][j]<p[j];
```

Figure (10)

Sample of constraint programming with CPLEX

4.2 Experimental Results and Discussions

As explained in Chapter 3, the problem was divided into two parts small size and big size. For each part different experiments were run.

4.2.1 Small Size problems

To better validate the experiment, three random data sets s_1, s_2, and s_3 are generated for every problem size. Two methods have been used to generate the random data: *Mersenne Twister Random Number Generator* and Java class*Random-number*. Every problem instance is solved by the three methods: MIQP, Constraint Programming and the new heuristic. Tables 11, 12, 13 and 14 summarize the results of C_{max} and the processing times of the three methods for the different problem instances.

Table (11)

Results of MIQP for limited size problems

# of jobs and machines		C_{max}			CPU Time (second)		
n	m	s_1	s_2	s_3	s_1	s_2	s_3
10	3	201	181	178	46	40	55
10	5	122	174	151	140	122	120
15	3	92	222	220	90	85	99
15	7	155	67	72	580	298	340
20	4	48	95	50	3720	3602	3640
20	8	82	100	191	1850	1914	1730
25	6	140	160	420	4187	4705	3831
25	10	59	105	231	7299	8364	10485
30	5	231	293	290	8583	10536	9750
30	10	-	-	-	>14400	>14400	>14400

Table (12)

Results of CP for limited size problems

# of jobs and machines		C_{max}			CPU Time (second)		
N	m	s_1	s_2	s_3	s_1	s_2	s_3
10	3	223	201	262	18	23	29
10	5	184	278	231	67	59	51
15	3	134	380	441	49	47	41.89
15	7	190	121	92	71	74	72.8
20	4	87	230	67	110	88	131
20	8	130	172	287	110	88	131
25	6	200	240	585	3668	3602	3612
25	10	101	190	300	4061	3990	3821
30	5	300	360	448	8440	9451	9224
30	10	338	-	-	13400	>14400	>14400

Table (13)

Results of the heuristic for limited size problems

when the machines are sorted

# of jobs and machines		A1			A2			A3		
		C_{max}			C_{max}			C_{max}		
		s_1	s_2	s_3	s_1	s_2	s_3	s_1	s_2	s_3
10	3	218	188	192	229	215	208	223	209	207
10	5	130	200	161	148	207	185	150	204	151
15	3	93	237	223	94	256	237	94	251	236
15	7	165	70	77	171	77	85	170	78	86
20	4	76	104	55	77	122	61	76	132	61
20	8	105	121	224	116	152	266	114	130	229
25	6	185	170	467	172	190	467	211	170	467
25	10	73	140	267	90	160	294	84	161	281
30	5	300	320	340	324	359	347	319	359	345
30	10	112	320	535	149	330	550	125	330	535

Table (14)

Results of the heuristic for limited size problems

when the machines are not sorted

# of jobs and machines		B1			B2			B3		
		C_{max}			C_{max}			C_{max}		
		s_1	s_2	s_3	s_1	s_2	s_3	s_1	s_2	s_3
10	3	218	188	192	229	215	208	223	209	207
10	5	130	200	161	148	207	185	150	185	151
15	3	93	237	223	95	267	237	95	260	236
15	7	165	70	77	171	77	86	170	77	89
20	4	69	115	55	77	121	62	76	104	61
20	8	105	121	224	116	152	229	114	131	229
25	6	184	161	467	206	161	503	208	190	426
25	10	74	132	259	90	160	290	84	161	280
30	5	300	320	340	324	359	347	319	339	344
30	10	112	310	535	149	330	550	125	330	535

For all tested instances, the Java program implementing the heuristic took less than 1 second to produce the results. To analyze and compare the makespan C^* produced by the MIQP (optimal) and the makespan $C^H_{\max}$ generated by the heuristic, a variable d_1 is introduced in the equation (4.1). The variable d_1 represents the percentage of the relative error of the makespan given by the heuristic and the optimal solution.

$$d_1 = \frac{C^H_{\max} - C^*}{C^*} \times 100 \qquad (4.1)$$

The results showing the makespan obtained by the MIQP and the new heuristic are shown in Table 15. This table records the best makespan obtained from the six different implementations A1, A2, A3, B1, B2, and B3.

Table (15)

Comparison between the MIQP and the heuristic

# of jobs and machines		s_1			s_2			s_3		
n	m	$C^H_{\max}$	C^*	d_1	$C^H_{\max}$	C^*	d_1	$C^H_{\max}$	C^*	d_1
10	3	218	201	8.5	188	181	3.9	192	178	7.8
10	5	130	122	6.5	185	174	6.3	151	151	0
15	3	93	92	1.1	237	222	6.7	223	220	1.3
15	7	165	155	6.5	70	67	4.4	77	72	6.9
20	4	69	48	43	104	95	10.1	55	50	10
20	8	105	82	28	121	100	21	224	191	17.2
25	6	184	140	31	161	160	0.6	426	420	1.4
25	10	73	59	23	132	105	25	259	231	12.1
30	5	300	231	29	320	293	9	340	290	17.2
30	10	112	-	-	310	-	-	535	-	-

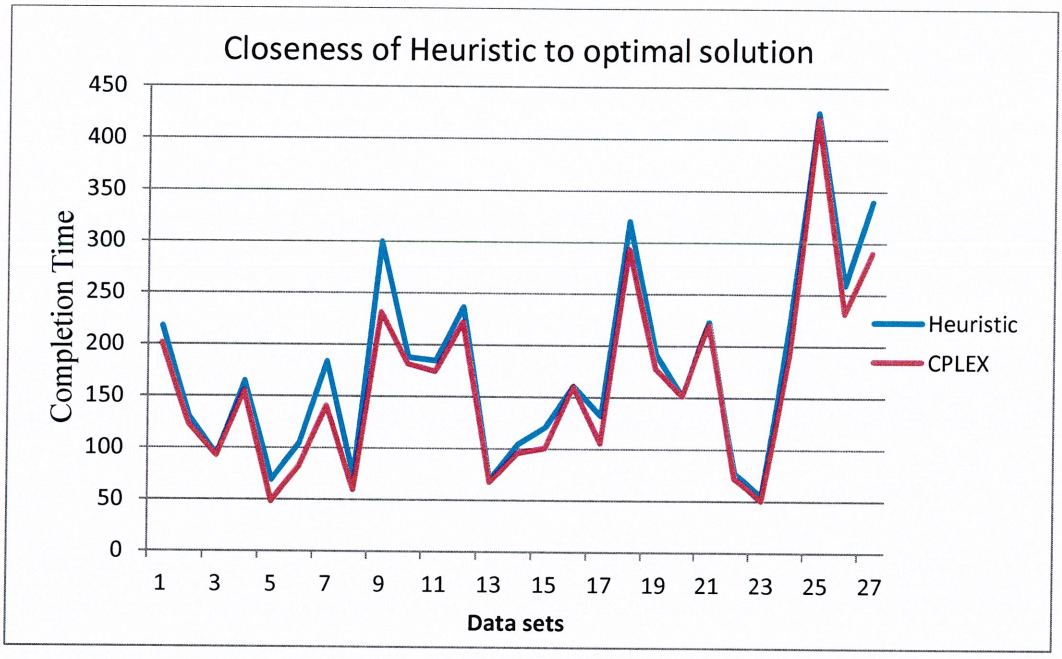


Figure (11)

Closeness of the of the solutions provided by the heuristic to the optimal solutions

We noticed that 95% of the results obtained by the heuristic for small size problems have an error less than 30% as shown in the Figure 11. The overall error average is 12.6%. The heuristic is able to find the optimal solution for some instances.

The same analysis is done to compare the performance of the CP to the MIQP results. A variable d_2 introduced in the equation (4.2) will represent the percentage of the relative error of the makespan given by CP $C_{\max}^{CP}$ and the optimal solution C^*.

$$d_2 = \frac{C_{\max}^{CP} - C^*}{C^*} \times 100 \qquad (4.2)$$

Table (16)

Comparison between the optimal solutions obtained by MIQP and makespan of CP

# of jobs and machines		s_1			s_2			s_3		
n	m	$C_{\max}^{CP}$	C^*	d_2	$C_{\max}^{CP}$	C^*	d_2	$C_{\max}^{CP}$	C^*	d_2
10	3	223	201	10.9%	201	181	11.0%	262	178	47.2%
10	5	184	122	50.8%	278	174	59.8%	231	151	53.0%
15	3	134	92	45.7%	380	222	71.2%	441	220	100.5%
15	7	190	155	22.6%	121	67	80.6%	92	72	27.8%
20	4	87	48	81.3%	230	95	142.1%	67	50	34.0%
20	8	130	82	58.5%	172	100	72.0%	287	191	50.3%
25	6	200	140	42.9%	240	160	50.0%	585	420	39.3%
25	10	101	59	71.2%	190	105	81.0%	300	231	29.9%
30	5	300	231	29.9%	360	293	22.9%	448	290	54.5%

As shown in Table 16 and Figure 12, the CP is not performing well for most of tested instances. The overall percentage error average is 53%. By considering the speed and the relative error, it is clear that the performance of the heuristic is much better than the Constraint Programming.

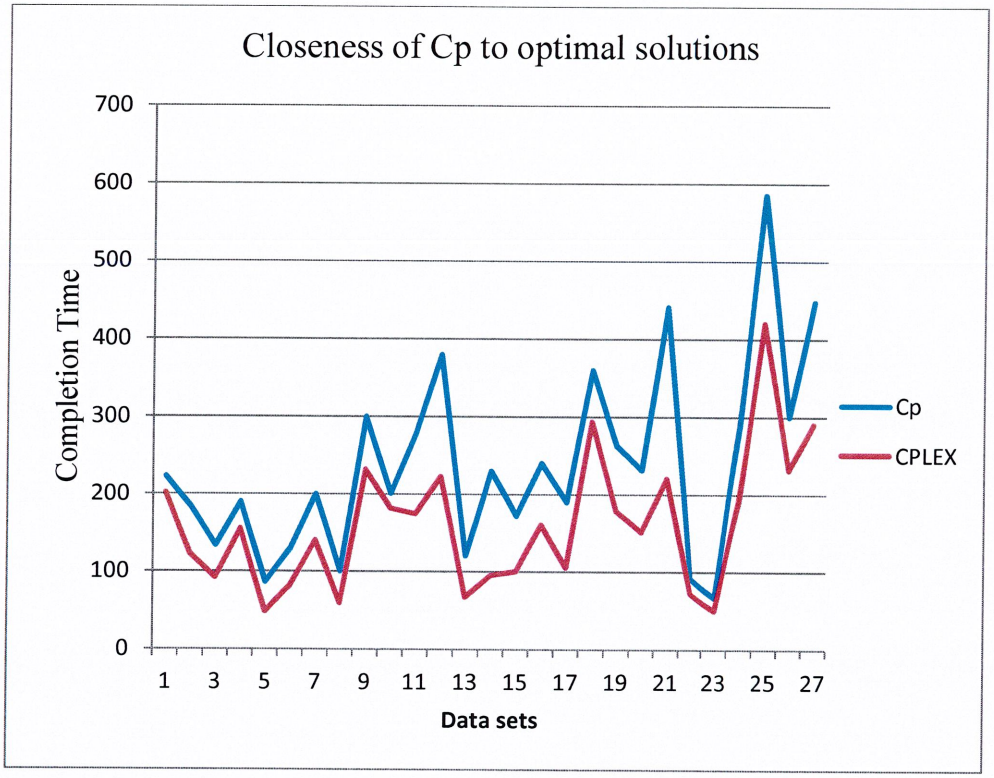


Figure (12)

Closeness of the solutions provided by CP to the optimal solutions

For each instance, the best solution obtained by the heuristic based on the different six setup stages is included in Table 17. In Table 17 $C^{H}_{\max}$ is defined as:

$$C^{H}_{\max} = \min(C^{H,A1}_{\max}, C^{H,A2}_{\max}, C^{H,A3}_{\max}, C^{H,B1}_{\max}, C^{H,B2}_{\max}, C^{H,B3}_{\max}).$$

Table (17)

Summary of the results and setup methods

# of jobs and machines		s_1		s_2		s_3	
n	m	$C^H_{\max}$	METHOD	$C^H_{\max}$	METHOD	$C^H_{\max}$	METHOD
10	3	218	A1/B1	188	A1/B1	192	B1/A1
10	5	130	A1/B1	185	B3	151	B3/A3
15	3	93	A1/B1	237	A1/B1	223	B1/A1
15	7	165	A1/B1	70	A1/B1	77	B1/B2
20	4	69	B1	104	A1/B3	55	B1/A1
20	8	105	A1/B1	121	A1/B1	224	B1/A1
25	6	184	B1	161	B1	426	B3
25	10	73	A1	132	B1	259	B1
30	5	300	A1/B1	320	A1/B1	340	B1/A1
30	10	112	A1/B1	310	A1/B1	535	A3/A1

The result shows that the LPT rule for both sorted and non-sorted machines (A1/B1) is most of the time providing the best makespan.

4.2.2 Big Size Problems

CPLEX cannot solve problems with more than 300 variables. In our case, 30 jobs and 5 machines was the maximum size that CPLEX was able to solve but taking almost 4 hours. To validate the performance of the heuristic, we compared its results to the range given in equation (3.6). An additional experiment was also conducted based on the Modified Graham Data Set.

Analysis using lower/upper bounds

In the first step, random data will be generated with two methods, *Mersenne Twister Random Number Generator* and *Random_Number*. Then, upper and lower bounds forming the range introduced before are calculated for each data set. To measure how far the makespan $C^H_{\max}$ generated by the heuristic for big size problems from the lower bound, the variable d_3 is proposed in the equation (4.3).

$$d_3 = \frac{(C^H_{\max} - lb)}{(ub - lb)} \times 100 \qquad (4.3)$$

Where $lb = (\sum_{j=1}^{n} p_j + \sum_{i=1}^{m} (e_i - s_i))/m$ and $ub = lb + \max_{j=1}^{n} p_j - 1$.

Table 18 includes the comparison results of 42 big-size problems. The results show that the heuristic is always generating makespan within the range. In addition, the results of 32 instances are less than 30% far from the lower bound as illustrated in Figure 13.

The experimental results show that the upper bound can be tightened to the average of job processing times as shown in the expression (4.4).

$$ub = lb + \sum_{j=1}^{n} p_j / n \qquad (4.4)$$

Table (18)

Upper/lower bound results

# of jobs and machines	Set	*lb*	*ub*	$C^H_{\max}$	$C^H_{\max} - lb$	d_3
$n=30, m=10$	s_1	169	239	184	15	21.4%
	s_2	109	177	117	8	11.8%
	s_3	36	46	47	1	10.0%
$n=30, m=8$	s_1	37	47	40	3	30.0%
	s_2	170	230	182	12	20.0%
	s_3	154.25	213.25	161	6.75	11.4%

$n=40, m=8$	s_1	48.17	105.17	61	12.83	22.5%
	s_2	193	249	197	4	7.1%
	s_3	353.5	441.5	359	5.5	6.3%
$n=40, m=12$	s_1	228.5	316.5	232	3.5	4.0%
	s_2	95.5	114	102	6.5	35.1%
	s_3	50.83	70.83	57	6.17	30.9%
$n=60, m=5$	s_1	190.1	210.1	191	0.9	4.5%
	s_2	975.2	1078.2	989	13.8	13.4%
	s_3	939.5	1037	942.8	3.3	3.4%
$n=60, m=12$	s_1	472.5	569.5	481	8.5	8.8%
	s_2	73.6	93.6	74	0.4	2.0%
	s_3	380.30	460.30	389.8	9.5	11.9%
$n=100, m=10$	s_1	765	838	774	9	12.3%
	s_2	315	336	318	3	14.3%
	s_3	577	651	581	4	5.4%
$n=100, m=20$	s_1	512	560	515	3	6.3%
	s_2	182.95	203.95	185	2.05	9.8%
	s_3	399.75	463.75	414	14.25	22.3%
$n=300, m=19$	s_1	1103.67	1167.67	1120	16.33	25.5%
	s_2	326.8	344.8	327	0.2	1.1%
	s_3	446.78	470.78	450	3.22	13.4%
$n=300, m=42$	s_1	291	316	295	4	16.0%
	s_2	231.78	256.78	238	6.22	24.9%
	s_3	883	981	914	31	31.6%
$n=500, m=21$	s_1	2519.04	2617.04	2547	27.96	28.5%
	s_2	1989.71	2068.71	2016	26.29	33.3%
	s_3	506.63	519.63	509	2.37	18.2%
$n=500, m=49$	s_1	193.16	205.16	198	4.84	40.3%
	s_2	546.89	592.89	558	11.11	24.2%
	s_3	1347.1	1474.1	1368	20.9	16.5%
$n=1000, m=60$	s_1	2212	2288	2263	51	67.1%
	s_2	1319	1444	1349	30	24.0%
	s_3	334	353	338	4	21.1%
$n=1000, m=105$	s_1	199.6	219.6	205	5.4	27.0%
	s_2	720	790	753	33	47.1%
	s_3	1805	1993	1868	63	33.5%

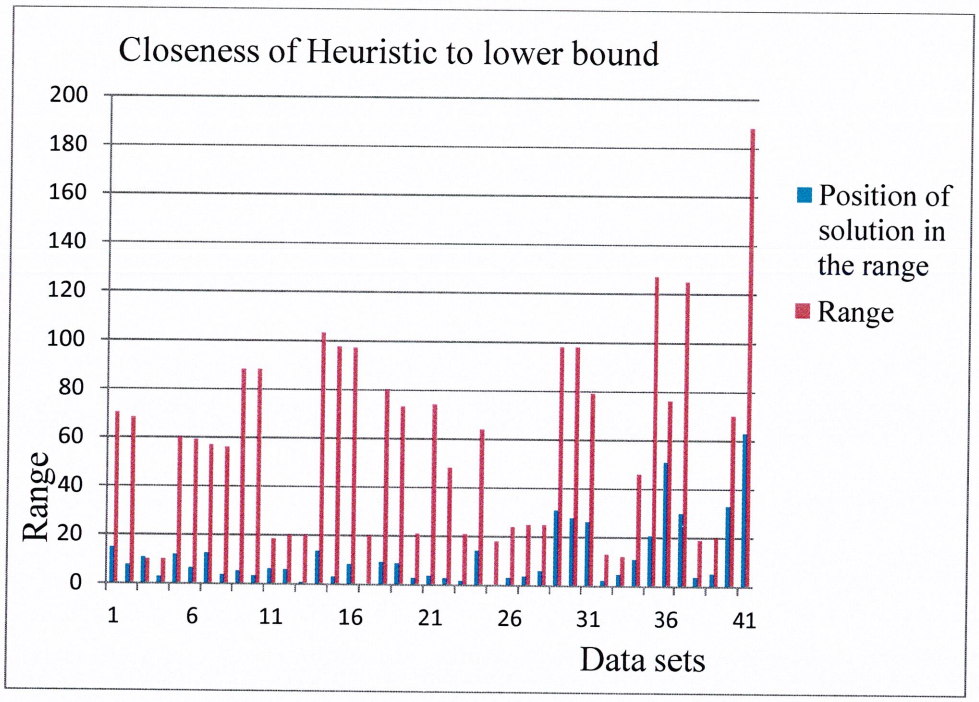


Figure (13)

Closeness of the heuristic results to the lower bound

Analysis using Modified Graham Data Set

In the second validation method, Graham Data Set was used to evaluate the heuristic's performance. Hashemian et al. (2014) used the Modified Graham Data Set to check their exact algorithm when machine unavailabilaty and operational preemtion are considered. They adapted the Graham Data Set by adding the unavailabilty periods for the machines as shown in Table 5. We used this method to compare the makespan $C_{\max}^{H}$ of the heuristic with the optimal makespan C^{*} produced by Hashemian's exact algorithm. The same variable d_1 given in the formula (4.1) is again used. As mentioned in Chapter 2, the Graham performance guarantee is less or equal to ρ_{Graham} given in the equation (4.5).

$$\rho_{Graham} = \frac{4}{3} - \frac{1}{3m} \tag{4.5}$$

One way to validate the heuristic's performance, is to calculate its approximation ratio using the formula (4.6).

$$\rho_{Heuristic} = \frac{C^{H}_{\max}}{C^{*}} \tag{4.6}$$

Table 19 summarizes the results of this experiment. The column labeled by d_1 shows that $C^{H}_{\max}$ is at most 40% far from the optimal solution. The Table 19 shows also that the heuristic approximation ratio is lower than Graham's approximation ratio. Which means the heuristic is performing very well compared to MGDS. This conclusion is more clear in the Figure 14.

Table (19)

Comparison between the heuristic results versus MGDS

# jobs and machines		C^{*}	$C^{H}_{\max}$	d_1	ρ_{Graham}	$\rho_{Heuristic}$
n	m					
61	30	105	129	22.9%	1.322	1.23
63	31	108	134	24.1%	1.323	1.24
65	32	111	140	26.1%	1.323	1.26
67	33	114	148	29.8%	1.323	1.30
69	34	117	147	25.6%	1.324	1.26
71	35	120	151	25.8%	1.324	1.26
81	40	135	160	18.5%	1.325	1.19
83	41	138	176	27.5%	1.325	1.28
85	42	141	179	27.0%	1.325	1.27
101	50	165	203	23.0%	1.327	1.23
109	54	176	214	21.6%	1.327	1.22
115	57	107	150	40.2%	1.327	1.40
121	60	200	228	14.0%	1.328	1.14
127	63	204	239	17.2%	1.328	1.17
133	66	220	260	18.2%	1.328	1.18
139	69	227	278	22.5%	1.329	1.22
145	72	280	280	0.0%	1.329	1.00
151	75	249	256	2.8%	1.329	1.03
157	78	252	350	38.9%	1.329	1.39
161	80	255	330	29.4%	1.329	1.29
201	100	414	575	38.9%	1.330	1.39
401	200	615	768	24.9%	1.332	1.25
601	300	915	1241	35.6%	1.332	1.36
801	400	1215	1572	29.4%	1.333	1.29

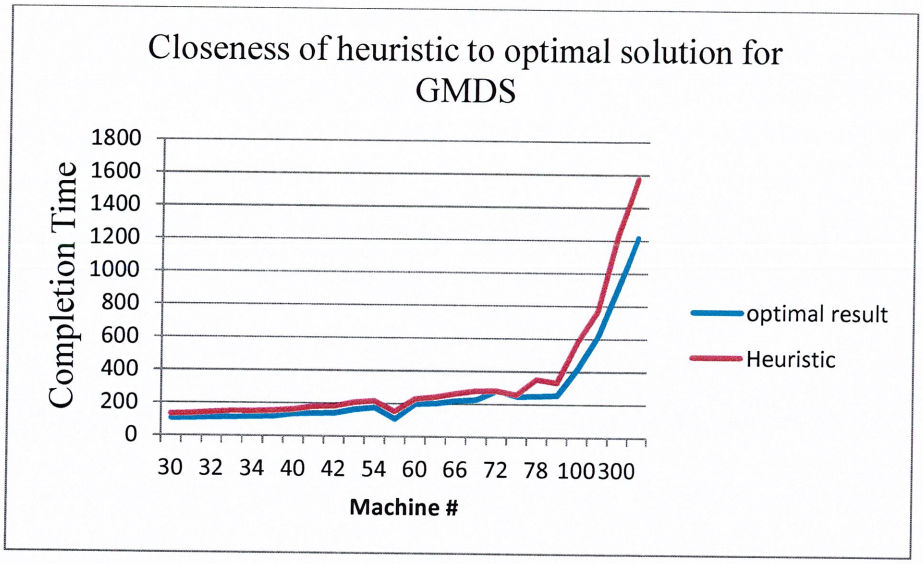


Figure (14)

Closeness of heuristic to optimal solution obtained from Modified Graham data set

4.2.3 Approximation Ratio

The relative approximation ratio of any approximation algoritm is defiend as:

$$\rho = OPT / f(x) \tag{4.7}$$

Where OPT is the value of the optimal solution and $f(x)$ the value of the solution obtained by an approximation algoritm.

The $\rho_{Heuristic}$ as defined in the formula (4.6) is calculated as below:

- For small size problems: Divides the heurisitc result by the optimal value obtained by MIQP.
- For big size problems:
  - Using bounds: The value of lb is considered to be OPT and $\rho_{Heuristic}$ is calculated by dividing the heurisitc result by lb.
  - Graham Data Set: $\rho_{Heuristic}$ is calculated by dividing the heurisitc result with the optimal value as shown in Table 19.

Table 20 is showing the average, maximum and minimum of the approximation ratio of the heuristic for the three cases we site before. The table shows there cases where the heuristic provides the optimal solution. In the worst case, the heuristic approximation ratio 1.43. The overall approximation ratio is 1.116667.

Table (20)

Summary of the relative performance gurantee of the heuristic

Problem		$\rho_{average}$	$\rho_{\max}$	$\rho_{\min}$
Small size probems		1.14	1.43	1
Big size problems	Using bounds	1.03	1.30	1.006
	MGDS	1.24	1.41	1

Chapter 5 : Conclusion and Future Work

5.1 Conclusion

5.2 Future Work

Chapter 5: Conclusion And Future Work

5.1 Conclusion

The scheduling problem of identical parallel machines under availability constraints has been studied. The problem is offline and arbitrary preemption is allowed where the jobs are resumable. The objective is to minimize the completion time. We showed that this problem is NP-Hard and as far as we know, no previous research has been conducted to solve it. However, there are many published works trying to solve closed scenarios.

A mathematical formulation as a Mixed Integer Quadratic Programming was proposed. Theoretical lower and upper bounds have been defined. In addition to that, the search space size of the problem has been set.

To solve the problem optimally, the MIQP was programmed and implemented using a CPLEX. This method is limited by the size of the tested instances. To overcome this limitation, a new heuristic was proposed. The heuristic uses priority rules such as LPT and SPT to assign the jobs as early as possible trying to avoid the preemption as much as possible. This heuristic tries to keep the machines busy during the planning horizon.

To validate the new heuristic many testing procedures were used. For small size problems, the heuristic results were compared with the MIQP. For big size problems, the developed bounds were used to check that the makespan provided by the heuristic is always in the accepted range. Another way to validate the heuristic is to compare it with the Modified Graham Data Set. The results obtained after all these validation

steps showed that the heuristic is performing very well for most of the tested instances. We showed that practically the theoretical upper bound can be tightened to smaller value.

5.2 Future Work

The experiment study showed for some instances high error ratios. We think this might happen because the range given by the lower and upper bounds is not tight enough. Developing better bounds will sure decrease the error ratio of the heuristic and therefore prove its high performance.

According to Sait and Sharif (2000), in order apply a local search method to solve a problem optimally; the search should start with a good feasible solution rather than a randomly constructed one. As the new proposed heuristic provides good feasible solutions for more than 79% of time, it can be used as seed for many other optimization methods like local search.

Another way to solve the problem can be investigated. In fact, we can adapt the methods developed to solve the subset optimization problem which is a particular case of the knapsack problem. These methods can be used to fill the available machine periods with jobs so that to eliminate the preemptions and therefore, solve the remaining tasks as $P_m \,||\, C_{\max}$ problem.

Reference

Baker, K. (2009). *Principles of sequencing and scheduling.* NJ: Wiley, Hoboken.

Bazaraa, M., & Sherali, H. (2008). *Nonlinear programming: Theory and algorithms.* Wiley.

Bazaraa, M., Jarvis, J., & Sherali. (2010). *Linear programming and network flows.* Wiley.

Blazewicz, J., & Drozdowski, M. (2000). Scheduling preemptable tasks on parallel processors with limited availability. *Parallel Computing*, 1195–1211.

Brucker, P. (2007). *Scheduling algorithms.* Springer.

Carlyle, W., Kim, B., & Fowler, J. (2001). Comparison of multiple objective genetic algorithms for parallel machine scheduling. *Evolutionary Multi-Criterion Optimization* (pp. 472-485). Mexico: Springer.

Chen, B., & Potts, C. (1998). A Review of machine scheduling: Complexity, algorithms and approximability. In P. M. Pardalos, *Handbook of combinatorial optimization* (pp. 21-169). Springer.

Chen, Z. L. (2004). Simultaneous job scheduling and resource allocation on parallel machine. *Annals of Operations Research*, 129, 135–153.

Constraint Programming Technology. (n.d.). Retrieved January 20, 2016, from IBM: http://www-01.ibm.com /software /commerce /optimization /constraint-programming-technology/

Dong, M. (2013). Parallel machine scheduling with limited controllable machine availability. *International Journal of Production Research*, 2240-2253.

Eren, T., & Guner, E. (2009). A bicriteria parallel machine scheduling with a learning effect of setup and removal times. *Applied Mathematical Modeling*, 1141-1150.

Frühwirth, T., & Abdennadher, S. (2013). *Essentials of constraint programming*. Springer Science & Business Media.

Gao, J., Guixia, H., & Wang, Y. (2009). A new parallel Genetic Algorithm for solving multi objective scheduling problem. *The International Journal of Advanced Manufacturing Technology*, 151-160.

Garey, M., & Johnson. (1978). Strong' NP-Completeness results: Motivation, examples, and implications. *Journal of the ACM*, 499-508.

Garey, M., & Johnson, D. (1979). *Computers and intractability: A guide to the theory of NP-Completeness*. W. H. Freeman.

Graham R, L. (1969). Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 1563–1581.

Graham, R. (1979). Optimization and approximation in deterministic sequencing and scheduling. *Annual Workshop on Computational Learning Theory and Discrete Math*, (pp. 287-326).

Han, B., Zhang, W., Lu, x., & Lin, y. (2015). Supply chain scheduling for parallel machine configuration with Single customer. *European Journal of Operational research*, 704-714.

Hashemian, N., Claver, D., & Béla, V. (2014). Makespan minimization for parallel machines scheduling. *Annals of Operations Research*, 173-186.

IBM ILOG OPL Language, R. (2010). *IBM ILOG OPL Language Reference Manual.* IBM.

IBM. (n.d.). *JAVA advantage*. Retrieved May 04, 2015, from IBM Knowledge Center: https://www.ibm.com/support/knowledgecenter

Kaabi, J., & Harrath, Y. (2014). A Survey of parallel machine scheduling under availability constraints. *International Journal of Computer And Information Technology.*

Kalantari, M., & Akbari, M. (2008). Fault Aware Grid Scheduling Using Performance Predication by Workload Modeling. *International Journal of Super Computing*, 15-39.

Kellerer, H. (1998). Algorithms for multi-Processor scheduling with machine release time. *IIE Transactions*, pp. 991-999.

Lee k., K., Leung, J. Y.-T., & Pinedo, M. L. (2013). Makespan minimization in online scheduling with machine eligibility. *Annals of Operational Research*, 189-222.

Lee, c. (1991). Parallel machine scheduling with non-simultaneous machine available time. *Discrete Applied Mathematics*, 53-61.

Lee, C.-Y. (1996). Machine scheduling with availability constraints. *Journal of Global Optimization*, 395-416.

Lei, L., Chung-Yee, & Pinedo, M. (1997). Current trends in deterministic scheduling. *Annals of Operations Research*, 1-41.

Lenstra, J., & Rinnooy, K. (1977). Complexity of machine scheduling problem. *Annals of Discrete Mathematics*, 342-362.

Leung, J., & Anderson, J. (2004). *Handbook of scheduling , algorithms, models and performance analysis.* United State of America: Chapman & Hal.

Liao, C., Shyur, D. L., & Lin, C. (2005). Makespan minimization for two parallel machines with an availability. *European Journal of Operational Research*, 445-456.

Liao, L., & Sheen, G. (2008). Parallel machine scheduling with machine availability and eligibility. *European Journal of operational research*, 458–467.

Lin, G., He, Y., Yao, Y., & Lu, H. (1997). Exact bound of modified LPT Algorithms applying to parallel scheduling with no simultaneous machine available time. *Applied Mathematics*, 109-116.

Matsumoto, M., & Nishimura, T. (1998). Mersenne Twister. *ACM Transactions and Modeling and Computer Simulation*, 3-30.

Pinedo, M. L. (2009). *Planning and scheduling in manufacturing and services.* NY: Springer.

Pinedo, M. L. (2012). *Scheduling theory, algorithms and systems.* Springer.

Rostami, M., Pilerood, A., & Mahdavi, M. (2015). Multi-Objective parallel machine scheduling problem with job deterioration and learning effect under fuzzy environment. *Computer & Industrial Engineering*, 206-215.

Saidy, H. R. (2008). Study of Scheduling Problems with Machine Availability Constraint. *Journal of Industrial and Systems Engineering*, 359-383.

Sait, S., & Sharif, Y. (2000). *Iterative computer algorithms with applications in engineering: Solving combinatorial optimization Problems.* Wiley-IEEE Computer Society Press.

Sanlaville, E., & Schmidt, G. (1998). Machine scheduling with availability constraints. *Journal of Global Optimization*, 395-416.

Schmidt, G. (2000). Scheduling with limited machine availability. *European Journal of Operational Research*, 1-15.

Tan, Z., & Young, H. (2002). Optimal on line Algorithm for scheduling on two Identical Machine with machine availability constrains. *Information Processing letters*, 89-93.

Wang, X., & Cheng, T. (2015). A heuristic for scheduling jobs on two identical parallel Machine with Availability Constrains. *International Journal of Production Economics*, 74-82.

Wu, & Lee. (2003). Scheduling linear deteriorating jobs to minimize makespan with an availability constraint on a single machine. *Information Processing Letters*, 89-93.

Wu, C., & Lee, W. (2007). A note on single machine availability with learning effect and availability constraints. *The International Journal of Advanced Manufacturing Technology*, 540-544.

Xu, J., Liu, C., Zhao, X. Z., & Sira, Y. (2010). Business process scheduling with resource availability constraints. *On the Move to*

Meaningful Internet Systems: OTM 2010 (pp. 419-427). Berlin: Springer.

List of Appendices

I. **Arabic Abstract**

ملخص

في فضاءات التصنيع كما في أماكن الخدمات العامة, تتعرض الآلات الى صيانة دورية مبرمجة مسبقا أو حتمية إثر حدوث أعتاب مفاجئة. يهتم هذا البحث بفئة معينة من آلات التصنيع و التي يطلق عليها الآلات المتوازية و التي تتشابه فيما بينها في الخصائص كسرعة الإنجاز و غيرها. يتمثل البحث الذي أجريناه في كيفية إنجاز مجموعة من الأشغال الغير مرتبطة ببعضها باستعمال مجموعة من الآلات المتوازية و التي تتعرض كل منها إلى فترة توقف مبرمجة مسبقا. يتمثل الهدف في إنجاز كل الأشغال بحيث يكون الانتهاء من آخر مهمة في أقصر وقت ممكن. ليكون مشكل البحث أقرب للواقع استعملنا افتراضات من مثيل إذا توقفت آلة ما عن العمل يجوز إنهاء ما تبقى من المهمة الحالية باستعمال أي آلة أخرى متاحة. قمنا بتصميم نموذج رياضي يصف المشكل المدروس باستعمال معادلات خطية و تربيعية. قدمنا ثلاث طرق مختلفة لحل المشكل أثبتت جميعها فعالية متميزة و ذلك باعتماد مشاكل نموذجية.

جامعة البحرين

كلية تقنية المعلومات



جدولة آلات موازية ومتطابقة مع أخذ الاعتبار عدم جاهزيتها في فترات معينه

أطروحة مقدمة كجزء من متطلبات الحصول علي درجة الماجستير في تقنية المعلومات

إعداد

الهام محمد اسكندرنيا

19927274

إشراف

د . يوسف حراث

(أستاذ مساعد)

جامعة البحرين

مملكة البحرين

يونيو ,2016