Department of Mathematics and Statistics

Task Scheduling in Parallel Processor Systems

Syarifah Zyurina Nordin

This thesis is presented for the Degree of

Doctor of Philosophy

of

Curtin University

November 2011

Declaration

To the best of my knowledge and belief this thesis contains no material previously published by any other person except where due acknowledgement has been made. This thesis contains no material which has been accepted for the award of any other degree or diploma in any university.

Syarifah Zyurina Nordin

Date : .....

Acknowledgments

Firstly, I would like to thank my supervisor, the Head of Department of Mathematics and Statistics Curtin University, Professor Louis Caccetta for his support and guidance throughout this study. All the research experiences with him helps me improved my study and research skills from time to time. I am very grateful to Ministry of Higher Education Malaysia (MOHE) and University of Technology Malaysia (UTM) for giving me the financial support and also the opportunity for me to study in Perth. I want to thank Curtin University especially the Mathematics and Statistics Department for being very supportive and very helpful whenever needed. Most importantly, to my beloved husband, Mohd Faiz Mat Daud and my two sons, Muhammad Aqiel Faris and Atieq Faisal, for their love, patience and sacrifices to me through all these difficult years. I also like to thank my parents, my parents in law and all my other family members for the encouragements and keep praying for my study. Last but not least, I would like to thank all of my friends for being concerned and helped me during my study especially to my former classmates, my colleagues from Mathematics Department UTM Skudai, my friends here in Perth and also of course my officemates in Mathematics Department, Curtin University. Thanks for making my life more joyful during this difficult journey.

Abstract

Task scheduling in parallel processing systems is one of the most challenging industrial problems. This problem typically arises in the manufacturing and service industries. The task scheduling problem is to determine a set of task assignments to a set of parallel processors for execution so as to optimize a specified performance measure. The difficulty of the problem is that the scheduling needs to satisfy a set of requirements as well as a range of environmental constraints. The problem is known to be NP-complete.

In this study, we consider a non-preemtive task scheduling problem on identical and unrelated parallel processor systems. We are interested in the objective function that minimizes the maximum of the completion time of the entire set of tasks (i.e makespan) so as to ensure a good load balance on the parallel processors. We consider three different task characteristics to the classical task scheduling problem that has a set of n independent tasks to be assigned to m parallel processors.

The first task characteristic that we consider is an on-line scheduling with release date specifications on an identical parallel processing system with a centralized queue and no splitting structure. We focus on developing simple and efficient heuristics for this problem. Three heuristic algorithms are proposed to solve this non-deterministic problem with scheduling over time where the availability of each task is restricted by release date. Our approach uses a multi-step method in the

task selection phase and a greedy search algorithm in the processor selection phase. The multi-step method is used to reduce the non-determinism in on-line scheduling by partitioning the scheduling process into several procedures. We introduce two procedures in the priority rule loop which we refer to as Cluster Insertion and Local Cluster Interchange. Computational testing on randomly generated data is conducted using Microsoft Visual C++ 6.0 to examine the effectiveness of the proposed multi-step method against the optimal solution. Different size of problems are tested in the experiment involving 3 processors by 200 tasks up to 5 processors by 1000 tasks with five clusters ranging from 10 to 50. The computational results show that all the three heuristics performed very well with the value of the average gaps are improved as the number of the tasks in the system is increases. The average gap for all the three heuristics are less than 1.04% for the largest tested cases (i.e for 1000 tasks run on 5 processors).

In the second problem, we address priority consideration as an added feature to the basic task characteristics of unrelated parallel processors scheduling. The priority consideration is defined by a list of ordered independent tasks with priority. A task requires to start processing after another task is finished on the same processor based on priority but may require to start earlier if processed on other machine. Our aim is to develop Mixed Integer Linear Programming models to obtain optimal solutions for three type of priority lists which are ascending order, descending order and general priority list. We validate the model using a case study taken from the literature. Then, computational testing is implemented on the general priority list using AIMMS 3.10 package and CPLEX 12.1 as the solver. Computational results show that the proposed MILP model is effective and produces optimal results for all tested cases. The model is very efficient as 95% of all the instances, which are problem up to 80 tasks assigned on 5 processors, have been solved within 5 minutes of CPU time.

In the final problem, we address a further problem for the task scheduling with a disruption problem that occurs on the parallel processor system. The disruption is causes by the unavailability of the processor during a certain time and it is called resource disruption. Our recovery solution for the disruption problem is a rescheduling approach. A MILP model is developed for the rescheduling model for the case of non-resumable tasks. Recovery model for the disrupted initial schedule with dummy insertion is proposed for predictive disruption management and match up schedule for post-disruption management. To evaluate the model, computational testing is performed with different sets of data. Different levels of disruptions are considered with different weights in the objective function to observe the stability of the model. The optimum initial schedule and the rescheduling model is performed using CPLEX 12.1 solver in AIMMS 3.10 package. In our computational results we measure the stability rate which is to observe the stability condition of the current schedule compared to the initial schedule in term of the task migration. From the results, the stability is improved when the number of tasks in the system increases within a reasonable amount of time. Another interesting observation is that our model yields small average gaps that are less than 7.99% within 300 seconds of the CPU time for a large data set that reach 200 tasks by 10 processors. The average gaps are considerably small for the disruption problem since the rescheduling model have to match up with the optimum initial schedule.

List of Publications and Presentations Related to This Thesis

1. Syarifah Zyurina Nordin and Lou Caccetta, A heuristic to minimize makespan in dynamic task scheduling on multiprocessors, *Proceedings of the 5th Asian Mathematical Conference(AMC2009)*, Kuala Lumpur, Malaysia, June 2009, 471–478.

2. Lou Caccetta and Syarifah Zyurina Nordin, MILP model in minimizing makespan with priority for unrelated parallel machines, *Proceedings of the International Conference of Optimal Control and Optimization (ICOCO2010)* , Guiyang, China, July 2010, 42–46.

3. Lou Caccetta and Syarifah Zyurina Nordin, Algorithm for task scheduling with priority in minimizing makespan for unrelated parallel processors, *The 8th International Conference on Optimization: Techniques and Applications*, Shanghai,China.

4. Lou Caccetta and Syarifah Zyurina Nordin, Mixed integer programming model for scheduling in unrelated parallel processor system with priority consideration, *submitted* .

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, parallel processing is one of the most rapidly growing technology due to the increasing demand in many areas of computer science and engineering. It has developed considerably over the past twenty years or so. Parallel processing is a form of information processing which uses concurrent events during execution. It is a simultaneous processing of the same task on two or more processors with the objective of completing the event within a given time.

Parallel processing can perform complex computations and reduce the time required to complete a certain problem. Therefore, parallel processing is particularly beneficial in various complicated applications such as manufacturing, space technology, weather forecasting, medical, military etc. In addition, scheduling is one of the most important issues in parallel processing applications.

In parallel processing, scheduling techniques are used to enhance the performance of the system and the scheduling problems have received an increasing level of attention recently. Scheduling is a method of assigning a number of tasks to process for processing. The scheduling program can be in serial or in parallel. In serial processing, the processes that occur run sequentially while in parallel processing they run in parallel. Therefore, scheduling in a parallel processing system takes less time to complete and is more effective especially for problems with a large volume of data.

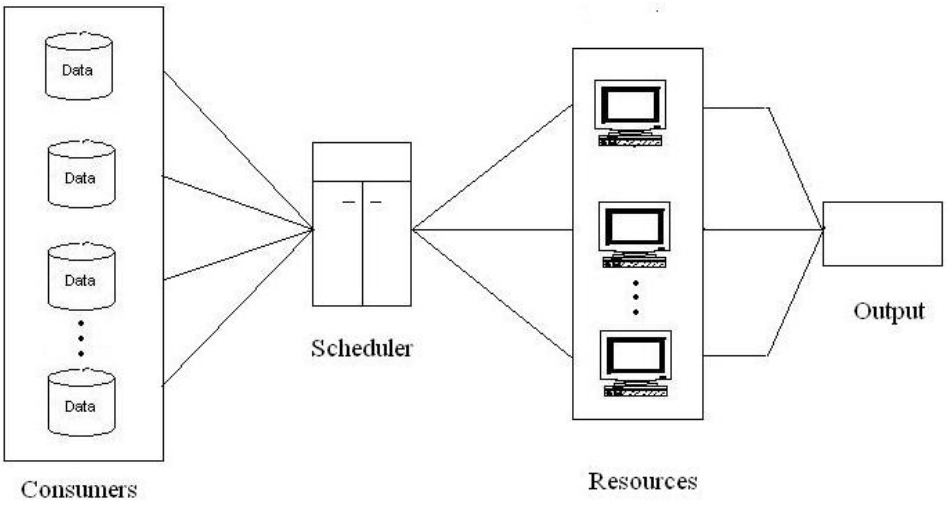

Figure 1.1: Scheduling in parallel processing system.

In general, the scheduling problem in a parallel processing system as shown in Figure 1.1 consists of a set of tasks that are normally considered as consumers that get processed by a set of parallel resources according to a certain policy . A real-life example that involves simple scheduling and parallel processing is in a busy shopping mall. The customers will form queues and wait for their turn at the check out counter. If the mall has got only one check out, then it may take a longer time for the customers to get served. However, if there are more than single counters, the customers can be split and will be served faster. This instance shows that scheduling by parallel processing is an effective solution for the real life problem. There are many other simple examples that involve scheduling in parallel processing systems such as banks, electronic manufacturing and aircraft maintenance process.

A scheduling system can be considered as consisting of a set of consumers, a set of resources and a scheduler. The example of the consumers are the customers in a shopping mall or in a bank, electronics devices in a factory and aircrafts at an airport. While the resources are the tellers at the check out counter or in a bank, factory operating machines and aircraft maintenance engineers. Other complex ex-

amples in recent applications in scheduling are in the area of printed circuit board (PCB) assembly process (Williams and Magazine, 2007), grid computing on meta-computers (Li, 2005) and radio access systems (Wan et al., 2003).

Cosnard and Trystram (1995); and Moldovan (1993) provided some applications on parallel processing based on technological developments that make a significant difference such as in engineering, material science, artificial intelligence and national defence area. In the national defence area, Moldovan (1993) mentioned that, Defence Advanced Research Project Agency (DARPA) was developed in United States in 1983 and initiated a program called Strategic Computing Program which involved parallel processing as the main technology. The scheduling system was one of the roles explored in the the Navy Battle Management (NBM) program. The NBM program was intended to deal with a complex situation in battle management especially during combat and crisis time. The program goal was to demonstrate a feasible system that is consistent and accurate.

Task scheduling is a challenging problem in parallel processing systems. In Sinnen (2007), task scheduling for parallel systems is refer to a schedule of task graph on a finite set of processors with start time and processor allocation functions. The purpose of the scheduling is to determine the allocation and the sequence of the operations to the target. A desired performance measure will be use to present the efficiency of the assignment. The task schedule needs to satisfy all the requirements and the constraints of the problem. A number of different types of parallel processors has been studied including: identical by Nordin and Caccetta (2009), Haouari and Gharbi (2004) and Mokotoff (2004); unrelated by Caccetta and Nordin (2010) and Ghirardi and Potts (2005); and uniform by Kravchenko and Werner (2009). There are also different task characteristics that have been considered such as due-date by Gordon et al. (2002a); preemtive by Cheng and Sin (1990); precedence constraint by Lenstra and Kan (1978); and on-line by Nordin and Caccetta (2009).

In this thesis, we consider problems with task scheduling on parallel processors systems. Our consumers are the tasks and our resources are the processors. Each task and processor have their own characteristics and specific environments need to be applied. The classification scheme of the tasks and the processors will be discussed later on. The term jobs and machines will be use exchangeably to the term tasks and processors. The scheduler generates a schedule and produces a timing diagram called Gantt chart as an output. The Gantt chart will illustrate the mapping of tasks to the allocated processors ordered by their processing time from the start to the finish time. Our objective is to produce a good performance of scheduling system with efficient policy. We are particularly interested to find a minimum of the maximum total execution time of the parallel processors as the performance criteria. Our aim is to develop mathematical models for scheduling problems and produce effective heuristic methods for obtaining nearly optimal solutions. We shall first introduce the basic scheduling notation and terminology for scheduling classification scheme in Section 1.1. The outline of the thesis is presented in Section 1.2.

1.1 Notation and Terminology

In this section, we provide some definitions and terminology used to describe the task scheduling problem throughout the thesis as there are variation of notation in the literature. The terminology and the notation are taken from Du and Pardalos (1998); El-Rewini et al. (1994);Cheng and Sin (1990); and Li and Yang (2009).

The scheduling problem that we consider is a system consisting of n independent tasks (or jobs) to be processed by m parallel processors (or machines). A **schedule** determines the allocation and the execution order of each tasks, J_i (where $i = 1, 2, \ldots, n$), on each processors, M_j (where $j = 1, 2, \ldots, m$). A **feasible** schedule is when the system satisfies the following conditions:

1. there is no overlapping of time interval corresponding to the same task,

2. there is no overlapping of the interval corresponding to the same processor,

3. all requirements of the problem type are satisfied.

Condition 1 ensures that each task in the system can only be assigned to one processor only at once and condition 2 means that each processor can process not more than one task at a time. The problem type mentioned in condition 3 is specified by processor environment, task characteristics and performance criterion.

Processor Environment

The processor environment is a system configuration of processors. There are different possible layouts of processors and we only assume the operation can be performs by only one type of processor. There will be no alternate scheduling. We also assume that all processors are ready to be served at time zero. All the processors are continuously available for production unless the condition of breakdown is considered.

There are three type of processor systems described by Cheng and Sin (1990): parallel processors and serial processors. In a **parallel processor** system, tasks can be receive as many as processors that are available at a time. Any tasks can also be assigned to any machine as each machine has the same function. However, there are three different categories of parallel processors which are identical, unrelated and uniform. The processors are said to be **identical** for the case that each machine has the same processing time for a job on any processor. The **uniform** machine scheduling problem is a natural generalization of the identical machine but each machine runs at a different speed. In the case that the m machine are **unrelated**, the processing time of a task can differ for each machine.

There are also other scheduling problem that can be solved with single or parallel processors named shop scheduling. The **shop scheduling** problem is a type

of multi-stage system. Each job requires operation at different stages with different functions. The shop scheduling problem also consists of a set of different processors that perform job operations. There are three types of systems in the problem which are called flow shop, job shop and open shop. A **flow shop** system has continuous layout with s stages. Each job follows the routing in order through stage $1, 2, 3, \ldots, s$. In **job shop**, there is no fixed routing among the jobs and each job has a different routing. Moreover, job shop has functional layout and all the jobs pass through the stages in batches. The **open shop** system has different processing where the jobs can enter and exit the system at any machine in which no restriction and no ordering constraints are placed on the operation. However, each processing job needs to go through each stage once and each of the job may have different routing.

A **serial processor** system is a layout that can be apply to flow-line type of production system. The serial processor has same-purpose machines and different-purpose machines. In a **same-purpose** machine, the machine can be identical or uniform type. The machine with identical category will function as aggregated unit and will increase productivity. However, the machine will reduce the flexibility of the system and increase the system reliability. Compare to a **different-purpose** serial machine, it can be apply to a continuous flow production as it trims the systems flexibility. Normally, this type of serial machine are used in the processing industry or system that have a large number of production.

In these thesis, we are not considered the shop scheduling problem and the serial processors. We only focus on the identical and unrelated parallel processor systems.

Task Characteristics

Every task has its own features based on the requirement of a system. We will describe the characteristics that are normally considered in parallel scheduling.

A **processing time** of a task can be denoted as p_{ij} is an execution time or a service time of tasks i when executed on processor j starting from the arrival time, a_i, until the finish time, C_i. We assume that all p_{ij}'s are greater than 0. In a parallel processing system, service time for identical machines is $p_{ij} = p_i$. The condition $p_{ia} = p_{ib}$ is for identical machines where a and b are different machines. Thus, a uniform machine has $p_{ij} = p_i/s_j$ where s_j is the speed of machine j and p_i is the processing time of job i at unit speed. In the case that the m machines are unrelated, $p_{ik} \neq p_{ir}$ for all jobs i and all machines $k \neq r$

Deadline or **due date** d_i is one of the types of time constraint which defines the latest time by which the execution of a task must finish. A process with a deadline can be divided into a soft deadline and a hard deadline. **Soft deadline** is a slight delay that exceed the completion time and missing the deadline can still be tolerated. On the other hand, **hard deadline** is a critical deadline that requires the task to be completed on time. If a slight delay happens, it may course a disaster effect to the system such as collapse or breakdown.

Release date (or ready time) r_i is the time at which the process is requested. Release date are used to ensure that the correct input is available to a process and that results are not delivered too early. In some situations, release date is same as the arrival time. **Arrival time** a_i is determine when it becomes known to the system. The arrival time of a task could be the earliest time at which the scheduler can begin to schedule the process. Situation $r_i = a_i$ happens when there is idle processor for the task at it's arrival time and is directly accepted for processing. In contrast, if the processors are all busy on the arrival time, the release date is the arrival time plus the delay time of the task.

In general, **preemtive** scheduling is a processing on any operation that permit a task to be interrupted. The task will be removed from the processor under the assumption that it will be resumed at a later time on the same or on a different processor. The task will still receive the execution time that it requires. With **non-preemtive** scheduling, a task execution must be completely done before another task is assign to the same processor, inferring that the job migration between processors is prohibited.

A **partial order** $\prec$ is a kind of dependent relation between tasks. The relation $T_i \prec T_k$ implies that T_i must be completed before T_k can begin on any processor. The pair of tasks T_i and T_k are called **dependent** tasks. The computation of T_k is depended on the computational result of T_i and the result of T_i must be known by the processor computing T_k. Partial order defined on dependent tasks specifies operational **precedence constraint**. For **independent** tasks, there are no precedence constraints and the set of the partial order is null. However, we defined a new restriction for the independent task and called it as **priority consideration**. Priority consideration applies the relation $T_i \prec T_k$ only on common processors. Moreover, T_k is allowed to start and finish before T_i in some circumstances on any other processor .

The availability of task information can be assumed to be either off-line or on-line. In a model which has **off-line scheduling** it is assumed that all the information of a task are ready and known before the execution starts. While **on-line scheduling**, in contrast to off-line scheduling, all the problem instances can be only known during the course of scheduling. The scheduling decision by the scheduler has to be made at execution time and in this model, the operation is **irrevocable**. All previous decisions to assign and schedule a job also may not be revoked. There are two ways the on-line task characteristics information is received by the scheduler: clairvoyant and non-clairvoyant scheduling.

In the **clairvoyant scheduling** model, it is assume that the scheduler will know all the requirements of a task after the task is either by scheduling over list or over time. The difference between scheduling over list and scheduling over time is the time when the scheduler exactly has knowledge about the task. In the case **scheduling over list**, the scheduler notices the job when all its predecessors have already been scheduled although all the jobs have already arrived and appear in a list. In the model **scheduling over time**, the scheduler knows the existence of a job only at their release date. While for **non-clairvoyant scheduling** model, the processing requirement of a job can only be known when the processing is finished.

There are special cases of the task characteristics in parallel processing systems where the system is disrupted. The **disruption** may occur at any time and there are two types of disruptions: machine disruption and task disruption. A **machine disruption** is when the processor becomes unavailable for a certain amount of time and the tasks that are assigned to the disrupted processor need either a another processor to gets serve during the disruption or wait until the machine get fixed and become available again. A **task disruption** refers to the change in some task parameter such as a are new requirement for task processing time and due date. As an example, if the due date of the task changes from d_i to $d_i - \delta$, this means that the task is disrupted with the due date accelerated by δ.

Performance Criterion

We can classify the main performance criteria or optimality criteria into three main groups: completion time, flow time and due-date.

The **completion time** criteria of a job can be denoted as C_i. Its refer to the finish time of a job i on a machine. Functions that can be achieve from the scheduling are total completion time $\sum C_i$ and mean completion time $\bar{C}_i = \sum C_i/n$.

Moreover, other criteria that can be performed involve functions with a positive integer weight w_i such as weighted completion time $\sum w_i C_i$ or makespan C_{max} which is the maximum of completion time to the entire set of jobs where $C_{max} = maximum\{C_1, C_2, \ldots, C_n\}$.

The **flow time** F_i of a job i is the time a job spends in the system. It is the difference between the completion time and the release time, $F_i = C_i - r_i$. Sometimes, we can find in the literature, the flow time is also referred to as response time, sojourn time or latency. Measures that are normally considered are total flow time $\sum F_i$, mean flow time $\bar{F_i} = \sum F_i/n$, weighted flow time $\sum w_i F_i$ and maximum flow time $F_{max} = maximum\{F_1, F_2, \ldots, F_n\}$.

All lateness L_i, tardiness T_i and earliness E_i involve **due-date** d_i. The **lateness** is when a job i is completed after its due-date and can be define as $L_i = C_i - d_i$. The **tardiness** is the maximum of the lateness and 0, $T_i = maximum\{C_i - d_i, 0\}$. On the other hand, **earliness** occurs if a job i commences before its due-date $E_i = maximum\{d_i - C_i, 0\}$ and the **unit penalty** $U_i = 1$ for the case $C_i > d_i$ and 0 if otherwise. The measures that involve total, mean, weighted and the maximization of the function also can be applied for lateness, tardiness and earliness functions.

In some situations, the scheduling requires more than one of these criteria to be considered. For example in Wu and Ji (2009), two performance measures were considered in the objective function for their scheduling model involving weighted tardiness and weighted makespan.

Three-field Representation

There is a representation scheme introduced by Graham et al. (1979) to describe the problem in three-field notation $\alpha|\beta|\gamma$. The three-field notation defines the problem type where α represents the processor environment, β indicates the task character-

istics and γ shows the performance criteria.

α field - Processor environment

We consider $\emptyset$ as an empty symbol and the α field can be divide in α_1, α_2 and α_3 groups. The following are the notation described for all the α's.

1. $\alpha_1 \in \{1, P, Q, R, F, O, J\}$, where
   (a) 1 : a single machine
   (b) P : identical parallel processor
   (c) Q : uniform parallel processor
   (d) R : unrelated parallel processor
   (e) O : an open shop
   (f) F : a flow shop
   (g) J : a job shop.
2. $\alpha_2 \in \{\emptyset, m, s\}$, where
   (a) $\emptyset$: the number of processors/stages is arbitrary
   (b) m : there is a fixed number m of processors
   (c) s : there is a fixed number s of stages.
3. $\alpha_3 \in \{\emptyset, (P\bar{m}), (P\bar{m}_1, \ldots, P\bar{m}_s), (P)\}$, where
   (a) $\emptyset$: a single stage, or several stages each with a single processor
   (b) $(P\bar{m})$: multi-stage with $\bar{m}$ identical parallel processors at each stage
   (c) $(P\bar{m}_1, \ldots, P\bar{m}_s)$: multi-stage with $\bar{m}_k$ identical parallel processors at stage k
   (d) (P) : multi-stage with an arbitrary number of identical parallel processors at each stage.

Note that, for a single machine problem, we will fixed the value of m in α_2 with 1 and other groups are $\emptyset$.

β field - Task characteristic

The next field is β field and determines characteristic of the tasks. The notation of $\beta \subseteq \{\beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6\}$ are as follows:

1. $\beta_1 \in \{\emptyset, on-line-list, on-line, on-line-list-nclv, on-line-nclv\}$, where
   (a) $\emptyset$: off-line scheduling
   (b) $on-line-list$: on-line scheduling over list
   (c) $on-line$: on-line scheduling over time
   (d) $on-line-list-nclv$: on-line non-clairvoyant scheduling over list
   (e) $on-line-nclv$: on-line non-clairvoyant scheduling over time.
2. $\beta_2 \in \{\emptyset, r_i\}$, where
   (a) $\emptyset$: no release date are specified
   (b) r_i : tasks have release date.
3. $\beta_3 \in \{\emptyset, d_i\}$, where
   (a) $\emptyset$: no deadlines are specified
   (b) d_i : tasks have deadlines.
4. $\beta_4 \in \{\emptyset, pmtn\}$, where
   (a) $\emptyset$: no preemtion is allowed
   (b) $pmtn$: operations of tasks may be preempted.
5. $\beta_5 \in \{\emptyset, intree, outtree, tree, chain, prec\}$, where

(a) $\emptyset$: no precedence constraints are specified

(b) $intree$: precedence constraints on jobs are defined by a rooted tree which has indegree at most one for each vertex

(c) $outtree$: precedence constraints on jobs are defined by a rooted tree which has outdegree at most one for each vertex

(d) $tree$: precedence constraints on jobs are defined by a rooted intree or outtree

(e) $chain$: precedence constraints on jobs are defined where each vertex has outdegree and indegree at most one

(f) $prec$: jobs have arbitrary precedence constraints.

6. $\beta_6 \in \{\emptyset, p_i = 1, p_{ij} = 1\}$, where

(a) $\emptyset$: processing times are arbitrary

(b) $p_i = 1$: all jobs in a single-stage system (i.e. each job consists of one operation only) have unit processing times

(c) $p_{ij} = 1$: all operations in a multi-stage system (i.e. each job consists of more than one operation) have unit processing times.

As an example, a job with $\beta_1 = on - line$ together with $\beta_2 = r_i$ indicates online scheduling over time as all jobs arrive at their release date. A job with $\beta_1 = on - line - nclv$ together with $\beta_2 = r_i$ represents on-line non-clairvoyant scheduling over time.

γ field - Performance criteria

In the last field, γ represents the performance criteria that minimizing the completion time, flow time and due-date function.

1. $\gamma_1 \in \{\sum C_i, \bar{C}_i, \sum w_i C_i, C_{max}, \sum f_i, f_{max}\}$

2. $\gamma_2 \in \{\sum F_i, \bar{F}_i, \sum w_i F_i, F_{max}\}$

3. $\gamma_3 \in \{\sum L_i, \bar{L}_i, \sum w_i L_i, L_{max}, \sum T_i, \bar{T}_i, \sum w_i T_i, T_{max}, \sum E_i, \bar{E}_i, \sum w_i E_i, E_{max}\}$

In some situations, we will refer the performance criteria by X and mention the specific criteria later on when the objective function involves more than one criteria or involves different criteria other than we have stated here.

Methodology

Scheduling problems are a special type of combinatorial optimization problem. An **optimization** problem is a set of instances with a pair (F, c) where F is any set of the domain of feasible points and c is the cost function with a mapping $c : F \rightarrow R^1$. Problems in optimization can be divided into two categories which are problems with continuous variables and problems with discrete variables. The optimization problem with discrete variables are known as **combinatorial optimization** problem.

An **algorithm** is a step-by-step procedure for solving a computational problem. For a given input x, it generates the correct output $f(x)$ after a finite number of steps. The **time complexity** of an algorithm expresses its worst-case time requirements, i.e., the total number of elementary operations, such as additions, multiplications and comparisons, for each possible problem instances as a function of the size of the instance. An algorithm is said to be **polynomial** or a polynomial (time) algorithm if its time complexity is bounded by a polynomial in input size. In time complexity theory, a problem can be classified into three classes which are P, NP and $NP - complete$. Informally, the class P is the class of decision problems for which there is a poly-time algorithm. Efficient computation (for a given problem) will be taken to be one whose runtime on any input of length n is bounded by a polynomial function in n. Let I_n denote all binary sequences in I of length n.

Definition 1.1 (**The class P** (Wigderson, 2006)) A function $f : I \rightarrow I$ is in the class P if there is an algorithm computing f and positive constants A, c, such that for every n and every $x \in I_n$ the algorithm computes $f(x)$ in at most An^c steps.

The class NP contains all properties C for which membership (namely statement of the form $x \in C$) have short, efficiently verifiable proofs. As before, we use polynomials to define both terms. A candidate proof y for the claim $x \in C$ must have length at most polynomial in the length of x. And the verification that y indeed proves this claim must be checkable in polynomial time. Finally, if $x \notin C$, no such y should exist.

Definition 1.2 (**The class NP** (Wigderson, 2006)) The set C is in the class NP is there is a function $V_C \in P$ and a constant k such that

1. If $x \in C$ then $\exists y$ with $|y| \leq |x|^k$ and $V_C(x, y) = 1$.
2. If $x \notin C$ then $\forall y$ we have $V_C(x, y) = 0$.

The **NP-complete** problems are regarded as hard problems. $NP - complete$ problem does not posses a polynomial algorithm. A problem X in NP is $NP - complete$ if any other problem in NP can be solved in polynomial time by an algorithm that makes a polynomial number of calls to a subroutine that solves problem X. The theory of NP-completeness provides a guideline on what type of scheduling problem. Most of the scheduling problems have been shown to be $NP - complete$ ((Du and Pardalos, 1998); (Lenstra and Rinnooy Kan, 1978);and (Du and Leung, 1989)). Refer to Wigderson (2006) for further explanation about these complexity class of problems.

The scheduling problem also can be formulated on a mathematical programming model. A general mathematical model of an optimization problem can be stated as:

$$\text{Min (or Max) } z = cx \tag{1.1}$$

subject to

$$Ax(\geq, = \text{ or } \leq)b \tag{1.2}$$

$$x \geq 0 \tag{1.3}$$

where A, b and c are a given matrices with size of $m \times n$, $m \times 1$ and $1 \times n$ respectively. The term z in equation (1.1) is an objective function that want to be minimized or maximized based on the equations (1.2) and (1.3) which are called **constraints**. The x is an unknown decision variable for the optimization problem.

The mathematical model can be classified as **linear programming** (LP) model when the objective function and all the constraints are linear. If a linear model with some (but not all) of the variables constrained to be integer then it is referred as **mixed integer linear programming** (MILP) model. The pure **integer linear programming** (ILP) model is for the case when all variables in the model are force to be integer.

Exact algorithms or complete algorithms are guaranteed to find for every finite size instance of combinatorial optimization problem an optimal solution in bounded time (Xhafa and Abraham, 2008). In scheduling problems, a schedule has to satisfy all the constraints and has the optimum value of the performance criterion. Exact solutions can be found by enumerative algorithms such as branch-and-bound and dynamic programming algorithms. The process of solving a problem using a **branch and bound** algorithms can be conveniently represented by a **search tree**. Each node of the search tree corresponds to a subset of a feasible solutions to a problem. A **branching rule** specifies how the feasible solutions at a node are partitioned into subsets, each corresponding to a descendent node of the search tree. A **dynamic programming** algorithm has the property that each partial solution is represented by a **state** to which a value (or cost) is assigned. As an example for minimization problem, when two or more partial solutions achieve that same state, one with the lowest value is retained, while the others are eliminated. It is necessary to define the states in such a way that this type of elimination is valid. For $NP - complete$

problem type, the computational results indicate the exact algorithms are capable of solving small instances only within a reasonable time. Therefore, in order to face the complexity of the problem, researcher prefer to focus on developing heuristics algorithm with fast and efficient method.

A **Heuristic** is an algorithm for efficiently, generating a "good" feasible. The optimal solution of the solution is not guaranteed. Heuristics algorithms are classified with three types as follows (Li and Yang, 2009): meta-heuristics, approximation algorithms and other heuristics . **Meta-heuristic** algorithms using techniques in solving problems based on local search technologies called iterative improvement. Simulated annealing, tabu search and genetic algorithm are examples of the meta-heuristics that are used frequently to solve scheduling problem. **Approximation algorithms** generates approximation solution to a scheduling problem with a performance guarantee, ρ. A $\rho-$approximation algorithm is a polynomial-time algorithm that produces a solution within ρ times the value of optimal solution. Other heuristics are referred to the techniques that do not have performance guarantees and do not use a local search method in solving problems. In general, the quality of the heuristic algorithms is determined by the performance and the efficiency of the algorithm.

Moreover, the performance of the algorithm can be compared with the optimal value called gap. For a problem with the objective function is to be minimized, the **gap** is defined as the difference of the performance value obtained by the heuristics above the optimum value and can be written as $(H - Opt)/Opt$ where H is the value obtain by the heuristics and Opt is the optimal solution. If the Opt is unknown because of the time complexity, the performance can be evaluated using $(H - LB)/LB$ and $(H - BH)/BH$ where LB is the lower bound and BH is the best heuristic among the heuristics. Other than performance value, the efficiency of the heuristics can be measure using the CPU time. The **CPU time** is the elapsed time for algorithms solve a problem. Normally, the CPU time for large scheduling

cases will take hours to solve it.

In this thesis, our approach in solving the task scheduling problem in a parallel processing system is to develop algorithms using heuristic and mathematical programming model. A heuristic algorithm is introduced for the scheduling problem with on-line task characteristic as no exact algorithm can be obtained unless with off-line feature. For the second and the third problem which are priority consideration and resource disruption characteristics respectively, we solve using mathematical programming model that we developed to obtain the optimal results.

1.2 Outline of the Thesis

In this thesis we address the task scheduling problem in parallel processing systems in finding fast and efficient schedules that minimize the makespan as the objective function. Our main contributions are the development, the implementation and the computational testing of a number of heuristics and the MILP models for solving the task scheduling problems on parallel processor systems with specific task characteristics. In this section, the outline of the thesis is briefly presented.

We provide some background of task scheduling problems on parallel processor systems in Chapter 2. In Section 2.1, we briefly present the literature of the scheduling components which are processor environments, performance criteria and task characteristics. In Section 2.2, the specified task characteristics which are on-line scheduling, priority consideration and disruption environment have been reviewed.

In Chapter 3, we consider the on-line scheduling problem with release date on the identical parallel processing system. The system has dynamic scheduling policies and the arrival task has different release times. Section 3.1 provides the idea

of the on-line scheduling in parallel processing systems. We also give the optimum model that can be used to solve the classical identical parallel processing problem. In Section 3.2, we develop three multi-step heuristic algorithms to solve the non-deterministic structure of the problem which we refer as HA 1, HA 2 and HA 3. We present the heuristics for obtaining the assignment of tasks to processors and illustrate how the heuristics work with an example. The computational experiments are conducted and the results are revealed in Section 3.3. The developed heuristics are implemented on 1000 random problems with 200 to 1000 independent tasks. The results show the heuristics improve the quality of the solutions as the system size increases in term of the average gap from the optimum value. For example, the largest instances of the problem which is 1000 tasks by 5 processors obtains the average gap approximately 0.33% (for HA 1), 0.06% (for HA 2) and 1.04% (for HA 3).

Chapter 4 introduces priority consideration as a new task characteristic in the unrelated parallel processing system. The priority consideration is an ordered priority list that contains a set of independent tasks. We describe in detail the description of the problem with a specific example in Section 4.1. Then, in Section 4.2 mixed integer linear programming (MILP) models are developed to obtain optimal solutions for the ascending, descending and priority list order. We validate, implement and analyze the results of the model in Section 4.3. We implement the MILP model using 28 combinations of tasks and processors and simulates 560 random instances. The number of tasks that have been used are 20, 30, 40, 50, 60, 80 and 100. From the computational results, the model is efficient and able to produce optimal results in solving the task scheduling problem with priority considerations. Besides that, we record the average CPU time for the MILP model. From the testing, the MILP model produce less computational time of optimal results within a reasonable time with an average less than 5 minutes for problems up to 80 tasks runs on 5 processors. If we consider all the tested instances, a large percentage of 94.8 has been solved optimally within this time.

Chapter 5 focuses on a disruption problem obtained from resource shortage. The resource disruption occurs on unrelated parallel processor system where one of the processor are unavailable at certain amount of time and the problem has been mentioned specifically in Section 5.1. The rescheduling MILP models for the initial schedule are presented for predictive disruption and post-disruption management are defined in Section 5.2. The models are the recovery option of the task scheduling problem when the parallel processor facing the unavailability with respect to the disruption during the process. In Section 5.3, we implement the computational experiments and discuss the results. We compute two types of data set which are small data set that contains number of tasks, $n = 20, 50$ and a larger data set with $n = 100, 200$. From the results, our rescheduling model obtains a high stability rate that can reach until 96% even we need to match up with optimum initial schedule. The model also very efficient with an average gap between 1.5% and 8% within the 300 seconds of the CPU time for large data set up to 200 tasks on 10 parallel processors.

In Chapter 6, we present some conclusions and suggestions for future works.

Chapter 2

Review of the Task Scheduling Problem in Parallel Processor Systems

The task scheduling problem in parallel processor systems is important in finding minimum (or maximum) performance criteria that will satisfy the task requirements. A parallel processor system is a generalization of a single processor and has a very important role in real-time applications, especially in industry. Different task features can be applied to the task scheduling problem for parallel processors and can be solved using various efficient techniques. Such techniques are for defining the allocation and completion of tasks in the shared parallel processor system.

This chapter provides a review of previous work on the task scheduling problem in parallel processor systems. Section 2.1 presents the task scheduling problem and covers the processor system, the objective function, and task characteristics. This section also discusses some approaches in solving the task scheduling problem on identical parallel processor and unrelated parallel processor systems. Section 2.2 provides specific cases of the characteristics of this task: on-line scheduling, priority consideration, and disruption environment. In the last section of this chapter, Section 2.3, is a summary of this review.

2.1 The Task Scheduling Problem

Task scheduling on a parallel processor system is the process of assigning tasks to a set of parallel processors. The goal of scheduling is to have a feasible schedule that satisfies all the processors availability, task constraints, and achieves the overall performance objective that has been made. It is known that the scheduling problem is NP-hard (Akyol and Bayhan, 2007, Yu et al., 2002). In this section, three important components of the task scheduling problem are briefly discussed. These include processor system, objective function and task characteristics.

2.1.1 The Processor System

Scheduling problems have been studied extensively for different types of processing systems. The system include: parallel processors, single processor and shop scheduling. Shop scheduling is a type of multi-stage machine problem, or in some parts of the literature it also known as a dedicated processor. A general review of the machine scheduling problem for single, parallel, and multi-stage systems can be found in Chen et al. (1998). This includes the complexity, algorithms, and approximability for deterministic machine scheduling. In addition, Blazewicz et al. (1991) and Li and Yang (2009) compiled a large number of mathematical programming formulations for parallel processor scheduling problems.

In the single processor scheduling problem the objective of the scheduling is to find a sequence of tasks on one processor that minimizes the objective function. For example, Zhao and Tang (2010) and Low et al. (2010) recently considered the scheduling problem on a single processor that minimizes the makespan. Zhao and Tang (2010) focussed on a single machine problem with an aging effect and proposed a polynomial time algorithm to solve it. Low et al. (2010) considered scheduling on a single machine with flexible maintenance issues and they proposed an easy first fit decreasing algorithm to solve the problem. The algorithm performs

well.

For the multi-stage processor problem, there are three possible layouts: job shop, flow shop, and open shop scheduling as described in Section 1.1. For example, Kachitvichyanukul and Sitthitham (2011) presented a two-stage genetic algorithm (GA) for solving the multi-objective job shop scheduling problem. The proposed GA has two stages in evolutionary process, and the authors showed that the method performs three times better than the literature method based on multi-stage genetic algorithm. Lochtefeld and Ciarallo (2011) proposed a multiple objective evolutionary algorithm for the same problem and showed that this proposed algorithm performed better than a single objective GA. The flexible job shop scheduling (FJS) has been studied by Wang et al. (2010), where this problem is an extension of the traditional job shop scheduling problem, more relevant to current market requirements.

In flow shop scheduling, Dhingra and Chandna (2010) considered the problem with a multi-objective scheduling and proposed new meta-heuristics using simulated annealing (SA), called hybrid SA (HSA). Six dispatching rules-based HSA were developed and tested on problem with up to 200 jobs and 20 machines. Solving multi-objectives for multi-stage scheduling has gained the attention of some researchers. Sun et al. (2011) presented a survey of the problem in which they summarized some heuristic, meta-heuristic, and hybrid procedures. Their approaches have been used for large and complex situations. Hybrid flow shop (HFS) is also another extension of the classical flow shop problem, which has multiple parallel machines in each stage. Ruiz and Vazquez-Rodriguez (2010) reviewed the methods that have been proposed to solve the HFS problem using either exact, heuristic, or meta-heuristic algorithms.

Huang and Lin (2011), Panahi and Tavakkoli-Moghaddam (2011), and Naderi et al. (2011) are among the researchers that have recently explored the open shop

scheduling problem, using different approaches. Huang and Lin (2011) proposed a bee colony optimization algorithm to reduce the solving time of the problem. Panahi and Tavakkoli-Moghaddam (2011) used another artificial intelligence approach, a novel hybrid ant colony optimization, to improve the quality of the schedules generated. Naderi et al. (2011) constructed the problem with an effective MILP model. Then, they proposed several meta-heuristic algorithms employing memetic and SA algorithms. The model and the algorithms showed good results in the experiments that were conducted.

On the other hand, task scheduling can also be performed on processors with different types of speeds. A parallel processor executing the task with speed s is called a uniform processor. Each task has identical processing time at the beginning but one must consider a consistent speed for each processor. The new processing time for each task has to be divided by the speed of the particular processor. If the processor runs with a high speed, the processing time of a task would be faster and the task would finish in a shorter time. In contrast, if the processor already has a lower speed, all the tasks assigned to the processor will take longer to complete. Different methods have been proposed on uniform parallel machines including: a branch and bound exact algorithm (Bekki and Azizoglu, 2008); meta-heuristic algorithms such as SA (Li et al. ,2011), GA (Zaidi et al., 2010) and an approximation algorithm (Fujimoto and Hagihara, 2006); and list-scheduling based algorithms such as the longest processing time (LPT) algorithm (Koulamas and Kyparisis, 2009, Lioa and Lin, 2003), and the earliest due date (EDD) algorithm (Koulamas and Kyparisis, 2000).

Other types of configurations in parallel systems are the identical parallel processors and the unrelated parallel processors. The uniform processor is the generalization of the identical processor. A processor is classified as an identical processor when the tasks are assumed to have a common processing time running on any of the processors. In this type of processor, the speed for every processor is always 1

or infinitely small. In an unrelated parallel processor system, every processor will receive a task with a different processing time. The processing time is not uniform among the processors. These two types of processors will be discussed further in Section 2.1.3 with specific applications and classical problems. The study of special features is made in Section 2.2 and the details are in Chapters 3, 4, and 5 of this thesis.

2.1.2 Objective Function

The objective function is a goal that needs to be achieved in the task scheduling to analyze the performance of the executing system. For due-date optimality criteria, Koulamas (2011), Tuong et al. (Tuong *et al.* (2009)) and Bilge et al. (2004) concentrated on minimizing the tardiness, and Toktas et al. (2004) studied an earliness objective function for their scheduling problem. For a comprehensive review of the due date type of problem we refer to Gordon et al. (2002b) and Lauff and Werner (2004). They surveyed scheduling with earliness and tardiness problems about a common due date on single and parallel processing systems. Such applications are for events that involve deadlines, for example a product which has to be completed and delivered on time. An application that involved this criterion was a pharmaceutical manufacturing plant (Ciavotta et al., 2009) and a discrete manufacturing enterprise (Xiuli et al., 2010).

Minimizing the flow time of task runs in the system is another objective function for the performance of the task scheduling problem. Flow time means the elapsed time for a task, from arrival until completion. Minimizing the total flow time has been studied widely for the parallel machine environment. For example, Chaudry (2010) considered scheduling with minimizing flow time on an identical parallel machine. Chandha et al. (2009) addressed the scheduling problem of minimizing the flow time on unrelated parallel machines with and developed an $O((1+\epsilon^{-1})^2)$-competitive algorithm where ϵ is refers to the speed. Kravchenko

and Werner, (2009) decided to apply scheduling to a uniform parallel machine environment, with the minimizing of the mean flow time as the objective function. In some applications, the flow time of a job correlates with the energy usage. Therefore, Lam et al. (2008) and Albers and Fujiwara (2007) considered minimizing the energy and total flow time as their scheduling objective.

The objective function for our model is based on completion time. We consider minimizing the makespan as our specific optimality criterion. Makespan is categorized as a min-max criterion as it refers to minimizing the maximum of task completion time in the system. There are a number of scheduling studies using makespan as the criterion for different types of machine environment, such as single machine (Parsa et al. (2010) and Low et al. (2010)), identical parallel processor (Kellerer (2008), Xu et al. (2008)), uniform parallel processor (Lin and Liao (2008), Liu et al. (2009)), and unrelated parallel processor (Rocha et al. (2008), Vallada and Ruiz (2011)). Makespan always relates to the utilization of the system. If the makespan is low, the utilization is high and hence, optimizes the performance of the system. It is also possible to consider other completion-time based criteria, such as minimizing the total completion time (Ji and Cheng (2008), Liu and Lu (2009)) and mean completion time (Allahverdi and Al-Anzi (2008), Tavakkoli-Moghaddam and Rahimi-Vahed (2007)). However, we only focus on one type of objective function in our model, makespan. Makespan is really important in a real system where it has the effect of load balancing over the parallel processors (Wotzlaw, 2007). For scheduling that involves more than one objective as performance measures, Lei (2009) presented an intensive review on bi-objective and multi-objective measures for scheduling performance.

2.1.3 Classical Task Characteristics

In the classical task scheduling problem in a parallel system, the model of the scheduling is classified as deterministic. Deterministic scheduling can be achieved

by the approach of off-line scheduling where all the information about the tasks and the processors is already known before execution. Standard scheduling also is restricted to non-preemptive scheduling. With non-preemptive scheduling, a task is not allowed to be interrupted and the task will be run on the same processor to completion. There are also no precedence constraints considered in basic scheduling. Therefore the task in the system is an independent task of which the computational results are not related to other tasks. The arrival time of all tasks is assumed to be at time 0 and after that, the tasks are available for processing. The processing time of the tasks in the system is also arbitrary, i.e., unequal from one another. The presence of deadlines is also ignored in this case.

Task scheduling problems with independent jobs on parallel machines in order to minimize the maximum completion time among the tasks, C_{max}, have received considerable attention from researchers for over a decade. Therefore, for the classical task scheduling problem, we consider minimizing the makespan as the objective function and this can be denoted by $P||C_{max}$ for an identical parallel processor and by $R||C_{max}$ for an unrelated parallel processor. Classical parallel processor scheduling has been studied extensively (Wotzlaw, 2007). The following sections are a survey of $P||C_{max}$ and $R||C_{max}$ scheduling problems.

Scheduling on Identical Parallel Processors

Scheduling on identical parallel processors is very important for real-time applications in many industrial applications all over the world. For example, the identical parallel processor has been applied in practical in manufacturing environment at a steel making factory in China (Xiaoguang, 2000) and also to pharmaceutical production plant manufacturing, located in Italy (Ciavotta at al., 2009). The identical parallel processor also can be found in the food industry. Doganis and Sarimveis (2008) considered a complex production process for industrial yogurt production in Greece. They have to optimize the yogurt packaging lines which are operated

by identical parallel automated packaging machines. This is a challenging problem and needs a very efficient production scheduling as the yogurt product has many features to take into account, such as yogurt culture, flavour, ingredients, label, cup size, etc. At the same time, the production has to fulfill the customers' demands and there also achieve the goal of minimizing production cost. Their approach has produced a customized MILP model for the problem to obtain the optimal schedule for the packaging lines.

Moreover, nowadays, most industrial companies prefer to use a modern approach by developing a computerized scheduling system rather than to find a schedule manually that might take a long time to generate. For example, the shipbuilding industry needs a very detailed manufacturing process especially for the block erection scheduling problem that was studied by Jinsong et al. (2009) using a computer search technique. They defined the block erection scheduling problem in shipbuilding as the identical parallel machine scheduling problem in minimizing the makespan. The identical parallel machine in this model represents the cranes at the dock which have to assemble the block on the docks. However, the cranes are limited and one needs to maximize the production as much as possible and minimize the makespan at the same time. They proposed a GA for the problem. The proposed GA yielded good approximation solutions with a 12% relative gap from the lower bound and a 10.6% percentage reduction in makespan.

The classical identical parallel machine $P||C_{max}$ for case $m \geq 2$ is NP-hard in a strong sense (Blazewicz et al., 1991). Therefore, an exact method for solving the problem is considered unlikely. Mokotoff (2004) proposed an exact cutting plane algorithm for the $P||C_{max}$. The algorithm consists of generating constraints for the polyhedral structure with valid inequalities and solving the linear program (LP) relaxation. The solution of the last solution from LP relaxation is used as an initial solution for the branch and bound procedure to continue the search. The algorithm is very effective when obtain optimal results in all small tested cases with

size problem in range of three machines by five tasks and five machines by 15 tasks. Amico et al. (2008) presented a branch and price scheme which is a generalization of the branch and bound (BB) procedure with LP relaxation. They solved the LP relaxation by a column generating algorithm at each decision node throughout the BB branching strategy.

Amico et al. (2008) also considered a meta-heuristic algorithm in solving $P||C_{max}$. They begin the method by computing the lower bound and the upper bound. Then, they improved the solution through a scatter search algorithm. The algorithm improved each solution by a local search and updated the feasible solution which they referred to as the reference set. The meta-heuristic algorithm is shown computationally to be an efficient method for solving a large number of instances. As for the approximation algorithm approach, Hochbaum and Shmoys (1987) are the standard reference for solving $P||C_{max}$. They obtained a polynomial-time approximation scheme using a dual approximation algorithm which they showed was less difficult than other approximation algorithms. They presented an algorithm with a running time of $O((n/\epsilon)^{1/\epsilon^2})$ for each ϵ and has a relative error of at most ϵ.

Much of the literature is concerned with developing other heuristic methods for solving $P||C_{max}$ and shows that the heuristic approach is promising in terms of the computation time and the size of the problem. Frangioni et al. (2004) are among the most effective heuristic algorithms for $P||C_{max}$. They proposed a new local search algorithm based on the multi-exchange neighbourhood. Alvim and Ribeiro (2004) derived an improvemed hybrid bin-packing heuristic for solving $P||C_{max}$. They combined their algorithm with the established longest processing time (LPT) algorithm that was proposed by Graham (1969). Gualtieri et al. (2008) also use the LPT algorithm in their proposed algorithm. They constructed a new $n\ log\ n$ algorithm to produce good solutions.

Scheduling on Unrelated Parallel Processors

The task scheduling problem on unrelated parallel machines has many applications in industry, and these have received an increasing amount of attention. For example, in Yu et al. (2002), the unrelated parallel processors scheduling problem considered arises in an actual printed wiring board (PWB) manufacturing line in the electronics industry specifically for drilling operations. All the lots are processed by a group of parallel flexible machines. Each machine has a different processing speed and operating characteristics. The processing time of each lot for drilling the PWBs may be different for different machines. As an example, machine A has a shorter processing time for lot 1 and may have a longer processing time for lot 2.

The unrelated machine scheduling problem also arises in semiconductor wafer manufacturing as reported by Kim et al. (2003), where the machines for dicing are categorized as unrelated processors. Their processing times vary depending on the ages and manufacture of the machines. Also, different setup times are required, depending on the job sequence, although the setup times are independent. This is because the sawing tools need to be changed for the dicing process, depending on the type of wafer. In addition, unrelated parallel machines may exist in stages due to the coexistence of new and old machines (Chen and Chen, 2009b).

The unrelated parallel processors problem not only occurs in manufacturing environments but also in the aircraft maintenance process (Kolen and Kroon, 1991). An aircraft must be inspected between the time of arrival and departure before being allowed to take off again. The aircraft maintenances processes are carried out by a number of ground engineers. However, a ground engineer is only allowed to do a specific maintenance job if they have a license for the corresponding aircraft type. Hence, operating job schedule needs to consider the maintenance jobs and the licences of the engineers. Each ground engineer is allowed to have only two licences at most. The scheduling problem considers the number of available engineers needed, in combination with the licenses to maintain the aircraft before take

off. Accordingly, the scheduling system can be considered as consisting of a set of unrelated parallel machines with corresponding operating characteristics and a set of jobs to be assigned to get a maximum total value.

The classical unrelated parallel processor scheduling problem on minimizing the makespan without additional task characteristics, $R||C_{max}$, has been widely studied using different approaches either exact or heuristic algorithms. An exact algorithm is an algorithm that finds an optimal solution. The schedule has to satisfy all the constraints and must have the optimum value of the performance criterion. Exact solutions can be found by enumerative algorithms such as branch and bound (BB)(e.g. Martello et al. (1997)) and dynamic programming algorithms (e.g. Horowitz and Sahni (1976)). The BB method can be represented as a search tree that produce a possible solution for each node of the search tree. Cutting plane schemes and branch and cut schemes have been developed in the literature as BB based methods. In dynamic programming, the procedure is to define and achieve the state with lowest value for the partial solutions while the other states are eliminated.

For an example of a BB based method, Mokotoff and Chretienne (2002) developed an exact cutting plane algorithm with polyhedral structure of $R||C_{max}$ problem. Basically, the separation procedure is used in the cutting plane algorithm for generating constraints. They have identified valid inequalities to allow them to develop an exact cutting plane scheme. In this scheme, the initial solution is obtained from the approximation algorithm of Davis and Jaffe (1981) and the lower bound is selected from the minimum processing time of each job. Then, the lower bound is improved using the LP relaxation introduced by Lawler and Labetoulle (1978). The new cut is derived from the linear program to decide whether the truncated polyhedral is empty or provides a feasible solution.

Martello at al. (1997) presented a branch and cut algorithm to solve the exact

$R||C_{max}$. They proposed a lower bound and also an additive bound on the Lagrangian relaxation and linear program (LP) relaxation. Then, they improved the bound with additional cuts that eliminate infeasibility in the cost function value. The results of the improvement bound is used in a BB algorithm and obtains the optimal result for $R||C_{max}$. Wotzlaw (2007) obtained a branch and price method in finding the exact solution of $R||C_{max}$ and this method is similar to the branch and cut method. The difference is that branch and price applies a column generation for solving the LP relaxation throughout the branch and bound tree.

Due to the complexity of unrelated parallel processor systems, considerable attention also has been focussed on the design of heuristic algorithms. Some researchers have developed metaheuristic approaches for the unrelated parallel processors problem, $R||C_{max}$. Glass et al. (1994) evaluated the comparison of the performance between a genetic algorithm and a neighbourhood search. The SA algorithm and Tabu Search (TS) performed well and provide good solutions compared to the genetic algorithm performance, which is likely to be poor. Srivastava (1998) developed an effective TS heuristic using a hashing approach to control the Tabu restrictions and diversification of the search process. Mokotoff and Jimeno (2002) proposed a new algorithm based on BB with an algorithm using partial enumeration and considering the integrality of a part of the set of binary variables. They concluded that this approach has less error than other metaheuristic algorithms that were developed before. Ghirardi and Potts (2005) also solved the $R||Cmax$ problem and use the Recovering Beam Search (RBS) algorithm approach which requires polynomial time. RBS is a method that enhances the traditional Beam Search (TBS) algorithm which is a truncated version of the BB algorithm. Recovering phase in the RBS allowed the algorithm to check the partial solution and recover from any wrong decisions occurring in TBS. They showed that RBS obtains good results on large instances within a reasonable time limit.

Many researchers use an approximation algorithm approach for the schedul-

ing problem. For example, Davis and Jaffe (1981), De and Morton (1980), Horowitz and Sahni (1976), Ibarra and Kim (1978) and Potts (1985) are among the earliest in using an approximation algorithm approach to solve the unrelated parallel processor problem $R||C_{max}$. Recently, Wotzlaw (2007) presented an approximation algorithm named the Approximation Unsplittable Truemper algorithm in solving the $R||C_{max}$ problem. The Approximation Unsplittable Truemper algorithm is a binary search method with 2-approximation factor within the best worst case running time.

In the literature, there many heuristic algorithms other than meta-heuristics and approximation algorithms. Chen and Chen (2009a) developed a bottleneck-based heuristic (BBFFL) to solve the unrelated parallel machine with a flexible flow line problem in minimizing the makespan. This heuristic consists of three steps: first, identify the bottleneck stage, second, generate an initial job sequence using the bottleneck-based initial sequence generator (BBISG), and third, generate the final schedule by applying a bottleneck-based multiple insertion procedure (BB-MIP). Lenstra et al. (1990) developed a heuristic cutting plane algorithm (HCPA) for the $R||C_{max}$ problem. They compared its performance with that of the exact cutting plane algorithm. The heuristic requires far less computation time than the exact due to the fact that its lower bound increases more rapidly. Fanjul-Peyro and Ruiz (2010) preferred a simple iterated greedy local search rather than a highly sophisticated procedure. They proposed seven new algorithms for $R||C_{max}$ and their proposed algorithm produced good solutions in a short computation time. in most situations, their methods improved the results obtained by Mokotoff and Jimeno (2002) and Ghirardi and Potts (2005).

In the next section, we will review some specific additional task characteristics in the classical task scheduling problem.

2.2 Specific Task Characteristics

2.2.1 On-line Scheduling

On-line scheduling refers to the availability of task information. On-line scheduling is in contrast to off-line and semi-online scheduling, where in on-line scheduling all the problem instances can be known only during the course of scheduling. While in off-line scheduling it is assumed that all the information concerning the tasks is known before the execution starts. Semi-online scheduling has some partial information about the job available before constructing a schedule (Tao et al., 2010).

In on-line scheduling, there are several classifications of possibilities on the arrival tasks. The most attention has been on on-line over list and on-line over time. On-line over list notices the task after all its predecessors have already been scheduled, and an on-line over time system knows the existence of the task at its release date. Therefore, the release date is one of the important task characteristic feature in on-line over time scheduling. The release date is a start time of the task available for processing or in on-line scheduling, the release date is exactly when the scheduler has knowledge about the task. In some conditions, the release date is the same as the arrival time when there is an idle processor for the task at its arrival time and directly accepts it for processing. In contrast, if processors are busy at the arrival time, the release date is the arrival time plus the delay time (i.e., the waiting time) of the task.

For a literature review survey about on-line scheduling, Sgall (1998) has presented comparative analysis results for different types of on-line paradigms. He mentioned in one theorem that for any variant of on-line scheduling of job arrival over time, there exist a 2-competitive algorithm with respect to makespan. Lee et al. (2010) provided a survey for on-line scheduling specifically in minimizing the makespan subject to machine eligibility. They proceed with the results for on-line over list and on-line over time. The review by Albers (2003) focussed on the com-

petitive analysis of the on-line algorithms.

Different algorithms have been obtained for solving on-line scheduling. For example, Hurink and Paulus (2008) showed, using a greedy algorithm, that on-line over list problems have a competitive ratio strictly not less than 2 . Fu et al. (2008) considered an on-line over time problem on unbounded batch machine to minimize makespan with limited restart. They provided an algorithm with a competitive ratio of $3/2$. Tian et al. (2009) proposed an on-line over time algorithm with a competitive ratio no greater than $\sqrt{2}$. They showed the bound is tight for the problem on two parallel batch machines with the objective of minimizing the makespan.

Some have researched on-line scheduling with other task characteristics. For example, Epstein (2000) addressed on-line scheduling with precedence constraints. The lower bound is proved to be $2 - \frac{1}{m}$ on the competitive ratio of any deterministic algorithm and $2 - \frac{2}{(1-m)}$ on any randomized algorithm. Both results are applied to preemptive or non preemptive cases. Huo et al. (2008) also considered precedence constraints and also other features with a set of equal-processing time with the objective of minimizing the makespan.

For multiprocessor systems, however, no on-line scheduling algorithm can be optimal in most cases (Funk, 2001). This is due to the lack of global information about the instances (Tao et al., 2010). Therefore, the optimum value for this on-line scheduling can be obtained when the scheduling is performed off-line.

2.2.2 Priority Consideration

A new task characteristic, that of priority consideration, in a parallel processor system is considered to give more choice than existing approaches in priority scheduling. In priority scheduling, the scheduling is assumed to be performed on-line and the task with higher priority than the other will be processed first (Xu and Parnas,

2000). The well known policies of priority scheduling are Rate Monotonic Scheduling (Liu and Layland, 1990) and Priority Ceiling Protocol (Sha et al., 1990). In our priority consideration task characteristic, we build an alternative, where the scheduling is performed in pre-run-time-scheduling where the schedule computes off-line. Xu and Parnas (2000) have discussed in detail the significant advantages of pre-run-time-scheduling compared to the priority scheduling scheme. We consider our priority consideration task characteristic with off-line scheduling so that we can provide the best solution in finding the optimum schedule.

The priority consideration task characteristic can be categorized as a partial precedence relation only. The system is not fully bounded by precedence constraints because this priority consideration has also an exclusion relation for the task that does not execute on the same processor. Priority consideration requires that a task k that has priority order after task i can only start on the same processor after task i is completed, but may be started earlier, on a different processor, before task i. This is different from a precedence constraint where task k is not allowed to start on any machine until task i completes.

In the literature, there are studies on scheduling that are focussed on precedence constraint on processor systems. Blazewicz and Kobler (2002) reviewed the scheduling problem with different properties of precedence constraints. For the single machine problem, Liu (2010) developed a BB algorithm to solve the scheduling problem with precedence constraint and release date in minimizing the maximum lateness. They used four different heuristics to find the upper bound and solved the relaxation problem for the lower bound. Gao et al. (2010) considered the same precedence constraint on a single machine but with start-time dependent processing time to minimize the total weighted completion time. They solved the precedence constraints of the jobs that are related by parallel chains and series parallel graphs. They proved the problems are polynomial solvable.

For precedence constraints on parallel processor systems, Ramachandra and Elmaghraby (2006) have presented a binary integer program and a dynamic program model for solving a two identical parallel processors system. They used a GA for the $m = 2$ and $m = 3$ identical processors problem. Omara and Arafa (2010) also developed a GA for mapping the precedence contraints task graph to the identical processors. They have proposed two improved GA and outperformed the traditional algorithm. They applied two fitness functions and a task duplication technique to improve the algorithm. Queyranne and Schulz (2006) worked on precedence delays for m identical parallel machines with release date and delivery time. A precedence delay is a certain amount of time that elapses between the completion and start times of a jobs. They provided a 4-approximation algorithm of list scheduling to minimize the total weighted completion time and improved the former approximation results of the problem. Kumar et al. (2009) and Liu and Yang (2011) considered precedence constraints on unrelated parallel machines. Kumar et al. (2009) obtained polylogarithmic approximations for precedence constraints which are tree-like, i.e., the undirected graph underlying the precedence is a forest. They have improved the bound for objective function in minimizing the weighted completion time and flow time. Liu and Yang (2011) employed a polynomial time algorithm, namely priority rule-based heuristic serial schedule (SS) algorithm. The computational experiment that they conducted showed that it is effective in minimizing the makespan with less computational time.

We consider our proposed priority consideration characteristic to be processed on an unrelated parallel processing system. When priority consideration is applied in the unrelated processor environment, the task k that is desired to be assigned earlier than task i on a different processor, has to take into consideration that the processing time is not the same. Therefore, the scheduler system for priority consideration on unrelated parallel processor has to take into account the priority order and the size of processing at the same time. This type of situation occurs in many applications in transportation such as berth allocation for cargo handling (Imai et

al., 2003). For the berth allocation problem, Imai et al. (2003) have to treat the vessel at various service priorities with different terminal space at the ports.

2.2.3 Disruption Problem

In classical scheduling, it is assumed that the system is always available for processing from the start time until the end. However, this assumption is invalid nowadays since parallel processing systems arising in many real applications especially in manufacturing applications with high levels of uncertainty, referring to the disruption that occurs in the system within a certain period. In practice, disruption in scheduling is one of the most important unexpected problem which are of concern. It is important to revise the schedule and overcome the disruption since it may cause a disaster or a significant loss of income if it involved a major disruption.

An extensive review of the literature on the disruption problem can be found in Vieira et al. (2003) and Aytug et al. (2005). Qi et al. (2006); Lee and Yu (2007); Goren and Sabuncuoglu (2010); Hall and Potts (2004); and Hall et al. (2007) studied the disruption problem on a single machine system. The same problem on parallel processor systems can be found in Alagoz and Azizoglu (2003); Azizoglu and Alagoz (2005); Tang and Zhang (2009); Ozlen and Azizoglu (2009); Ozlen and Azizoglu (2011); and Arnaout and Rabadi (2008). For a multi-stage environment, Rangsaritratsame et al. (2004) and Moratori et al. (2008) studied the disruption specified on job shop scheduling.

In Zhu et al. (2005), the causes of disruption were divided into three different types: project network, activities, and resources. Project network disruption is when the system is disrupted by new activities or changes in precedence relations of the project. An example of such a new activity is a change order from the customers either to order more or cancel an order. This situation happens in a case where an initial schedule has been made with all preparation and is then disrupted by new

orders unexpectedly before or during the execution of the schedule. Hall and Potts (2004) have considered project network disruptions specifically on new orders. The new order is limited to a single arrival and assigned in a single machine environment. They developed models to minimize the scheduling cost and the degree of disruption. Hall et al. (2007) continued with the same type of disruption problem but the single machine system processed the disruption in more general cases with multiple new orders arriving. They designed an approximation algorithm and proposed a branch and bound algorithm to solve the problem. The objective that they considered is to minimize the maximum lateness of the jobs. Moratori et al. (2008) investigated the disruption problem involving a new rush order and subject to precedence constraints. In the scheduling, the rush order needs to be accommodated as soon as possible. They presented their strategy using a match-up algorithm to solve the disruption problem for a job shop scheduling system.

Activity disruption occurs when there is a change in the parameter of the activity. Normally, the activity refers to a job of the scheduling. For example, a job is disrupted if the duration or the processing time is different from the initial plan. The job duration can either increase or decrease from the original schedule. Qi et al. (2006) and Goren and Sabuncuoglu (2010) discussed job disruption for single machine scheduling. In Qi et al. (2006), they solved the problem job disruption by a left-shift shortest processing time (LS-SPT) algorithm, and proved that approach is optimal for the disruption problem in minimizing the total completion time and the total tardiness. Goren and Sabuncuoglu (2010) also developed an exact algorithm for job disruption using the smallest variance of processing time first (SVPT), which method is optimal when machine breakdown is not being considered at the same time.

Resource disruption is a disruption that involves a shortage in resources such as machines and personnel. This disruption causes a schedule to become infeasible, as the resource is not available at the specified time. Alagoz and Azizoglu

(2003); Azizoglu and Alagoz (2005); and Tang and Zhang (Tang and Zhang (2009)) are among the researchers that have focussed on machine disruption. They applied the disruption problem in identical parallel processor environment. Alagoz and Azizoglu (2003); and Azizoglu and Alagoz (2005) considered the same machine disruption problem and they proposed several heuristic algorithms including a polynomial time algorithm and BB based heuristic to minimize the number of disrupted jobs and the total flow time. Tang and Zhang (2009) investigated machine breakdown disruption that can minimize the total weighted completion time and the weighted deviation completion time cost from the original schedule. They used Langragian relaxation to solve their model and obtained near-optimal solutions for all instances.

In this research, rescheduling for resource disruption is considered particularly for unrelated parallel processing systems and it is considered to be an NP hard problem (Ozlen and Azizoglu, 2009). Our problem is similar to the research addressed by Arnaout and Rabadi (2008); Ozlen and Azizoglu (2009); and Ozlen and Azizoglu (2011). Currently, to the best of our knowledge, there is nothing in the existing literature that has introduced this matter except for these three research contributions: Ozlen and Azizoglu (2009); Ozlen and Azizoglu (2011); and Arnaout and Rabadi (2008). Arnaout and Rabadi (2008) defined different rules in solving the machine breakdown problem: right shift repair, fit job repair, partial rescheduling, and complete rescheduling. They considered the problem of having idle time between the tasks during execution on the machine whereas we consider a machine with no idle time between the task. In Ozlen and Azizoglu (2009) and Ozlen and Azizoglu (2011), they proposed a BB algorithm and an exponential time algorithm, respectively, to solve the machine disruption problem in minimizing the total flow time and total disruption cost. We continue the disruption problem on unrelated parallel processor problem with makespan as the performance measure.

2.3 Summary

This chapter discusses some related work on the task scheduling problem. The different properties in the scheduling area from other researchers provide directions for further exploration. Therefore, the additional features are considered in the classical scheduling problem. In the next chapter, we will present the first feature, which has non-deterministic information on the in-coming tasks on an identical parallel processor system.

Chapter 3

On-line Scheduling in Parallel Processor System

This chapter studies the non-deterministic aspect in identical parallel processor systems. An added feature is considered to the standard task characteristic which is the on-line scheduling with different task release dates. On-line scheduling refers to the availability of task information. All the problem instances can only be known during the course of scheduling. On-line scheduling is in contrast to off-line scheduling where all the information of the tasks are ready and known before the execution starts. Heuristic algorithms are developed as the optimum solution for the problem can be obtain only for the off-line model. Our proposed heuristics for on-line scheduling problem are based on the multi-step algorithm and will be applied in the task selection phase.

This chapter consists of four sections organized as follows. In Section3.1, we describe the specific on-line scheduling problem and some related assumptions. Section 3.2 discusses the detail of our heuristics scheme. Section 3.3 provides the computational results. Lastly, we summarize the chapter in Section 3.4.

3.1 Problem Descriptions

The parallel processor system in this chapter can be stated as follows: We are given a set of n tasks, $J_i = \{J_1, J_2, \ldots, J_n\}$, that are to be processed on a set of m parallel processors, $M_j = \{M_1, M_2, \ldots, M_m\}$. The tasks are independent in that there are no precedence relations among them. The processors are said to be identical, when the processing time $p_{ij} = p_i$ for all tasks i and processor j with the same processor speed. The performance criterion of interest is the makespan ,C_{max}, of the system. In real-time systems, it is important to minimize the makespan, i.e the maximum of total tasks execution times for every processor, to keep balance the capacity utilization among parallel processors.

We are concern with scheduling policies in parallel processing systems with centralized and no splitting system structure as shown in Figure 3.1. In such systems, there are $m > 1$ identical processors. A set of sequence of tasks are arriving in the system at random points in time with arbitrarily distributed inter-arrival times. When a task arrives into the system, it is immediately queued up in order and centralized (i.e the queue is common for all the m processors). Each task requires an amount of random service time. The scheduling of tasks on the processors has no splitting structure since the entire set of tasks are scheduled to run sequentially on the same processor once the task are scheduled.

In this scheduling problem, we assume that all processors are available to serve at time zero and can process not more than one task at a time. Each processor is continuously available and there is no idle time between the execution of a pair of tasks. Each task has a processing requirement with positive processing time units. Each arrival tasks will be assigned to one processor only, inferring that the task migration between processors is prohibited. The processing of any operation may not be interrupted. Every task execution must be completely done before another task assigned to the same processor and each task must be successfully assigned to one

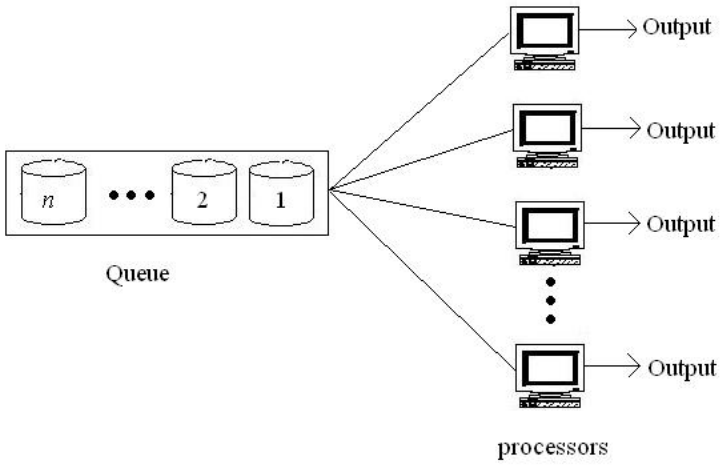

Figure 3.1: Parallel processing model-centralized and no splitting system

processor only. Moreover, each task i has a nonnegative integer release date, r_i, at which time it is becomes available for processing.

We consider the scheduling with on-line characteristic where all information is not known in advance. The scheduling decision by the scheduler has to be made at execution time and in this model, the operation is irrevocable. All previous decisions to assign and schedule a task also may not be revoked. In on-line scheduling, there are several classifications of possibilities on the arrival tasks. The most attention on on-line model systems has been focussed on on-line over list and on-line over time scheduling. On-line over list systems do not notice the existence of the arrival task until all its predecessors have already been scheduled while on-line over time systems know the existence of the tasks at their release date. The way of task characteristics information received by our scheduler is the model of scheduling on-line over time. Note that, for the multiprocessor systems, there are no on-line scheduling algorithms that can be optimal in most cases (Funk et al., 2001). It is due to the lack of global information on the instances (Tao et al., 2010). The optimum value for this on-line scheduling with task release dates can only be obtain when the scheduling is performed off-line.

From all these descriptions and assumptions, our task scheduling problem can specifically be denoted based on the established three-filed notation as $P|on-line, r_i|C_{max}$, i.e. the task scheduling problem for minimizing the makespan on identical parallel processors with the model requires scheduling to be on-line over time where the availability of each task is restricted by release date. We have added a substantial degree of difficulty to the classical problems $P||C_{max}$ which is already strongly NP-hard (Du and Pardalos, 1998).

The problem $P|on-line, r_i|C_{max}$ also has many applications in industry. For example, identical parallel machine scheduling systems can be found in real-world manufacturing environment such as in the integrated-circuit packing manufacturing particularly in wirebonding workstation (Yang, 2009). In wirebonding process, the system has jobs waiting for scheduling to process on a large number of parallel wirebonding machines. The operator will keep at least one load get onto a built in machine buffer for each wirebonding machines. The wirebonding operations also deals with the on-line job arrival case and minimizing the makespan as the objective. On-line scheduling on identical parallel processors also can be applied in another application involving high tech equipment on operating rooms in health care industry (Vairaktarakis and Cai, 2003). A utilization schedule needs to be prepared daily as well as monthly to fit the room with patients' cases (as many as possible) and facilities needed for each. The room scheduler will consider the operating room as the identical machine and the cases represent the jobs. The room utilization or the makespan is very critical to profitability.

We now provide, for further information on the $P|on-line, r_i|C_{max}$ problem. The optimum model for $P|on-line, r_i|C_{max}$ can be obtained from the mixed integer linear programming (MILP) model for $P||C_{max}$. We assume that the $P||C_{max}$ problem is deterministic with all the tasks arriving at time zero and all information regarding the tasks known beforehand. The model for the problem $P||C_{max}$ can be formulated as follows (Funk et al., 2001):

Let

$$x_{ij} = \begin{cases} 1, & \text{if task } i \text{ is assigned to machine } j \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

where task $i = 1, 2, \ldots, n$ and x_{ij} are the assignment variables. The MILP model is:

$Minimize \qquad C_{max}$

$s.t$

$$\sum_{j=1}^{m} x_{ij} = 1 \qquad \text{for } i = 1, 2, \ldots, n \tag{3.2}$$

$$\sum_{i=1}^{n} p_i x_{ij} \leq C_{max} \qquad \text{for } j = 1, 2, \ldots, m \tag{3.3}$$

$$C_{max} \geq 0 \tag{3.4}$$

$$x_{ij} \in \{0, 1\} \qquad \text{for } i = 1, 2, \ldots, n \qquad j = 1, 2, \ldots, m \tag{3.5}$$

In the objective function, C_{max} is the makespan that needs to be minimized. To ensure a feasible schedule, several constraints are required. Constraints (3.2) ensure that each tasks is assigned to only one machine. Constraints (3.3) guarantee that the total execution time for every task on machine j is less than or equal to the objective function.

In the next section, we will provide the proposed multi-steps heuristic algorithm and the computational experiments for the $P|on - line, r_i|C_{max}$ that have been published in Nordin and Caccetta (2009).

```
Procedure Overall_Algorithm()
  begin
      Initialization parameter;
      while (termination criterion not satisfied) do
          Local_Priority_Rule();
              while (Local_Priority_Rule) do
                  Cluster Insertion();
                  Local Cluster Interchange();
              endwhile
          Processor_Selection();
      endwhile
  end
```

Figure 3.2: Pseudo-algorithm for proposed heuristic

3.2 Proposed Heuristics Algorithm for Online Scheduling

In this section, we consider three heuristic algorithms that required less arrangement time for solving $P|on-line, r_i|C_{max}$ which are simple and fast. The fast arrangement time is needed as for the on-line scheduling, we have to compete with the running time. The proposed algorithms that we are going to introduce share a common structure. First, task selection phase is obtained with a loop of multi-step procedures. The proposed heuristics design a slot for a refinement step during the running time in the loop and produce new Q_{list} for each refinement step. At the first step of the loop, we apply priority rule for the whole system until the solution is obtained. Along with this step, two subsequent modification methods are applied but still in the priority rule loop. This modification steps are based on a Cluster Insertion (CI) and a Local Cluster Interchange (LCI) methods. After obtain a list of candidate in the task selection phase, the procedure continues with processor selection phase until the system is terminated. The overall structure of the proposed heuristics is given in Figure 3.2.

3.2.1 Task Selection Scheme

Local Priority Rule

A priority rule in task scheduling is a policy in assigning an available task to an idle processor with a specific sequencing decision (Haupt, 1989). An available task refers to a waiting task that is ready to get assign. The goal of applying a priority based rule is to select the task which has to be scheduled next, so that the system correctly allocates the resource. The rule also gives acceptable results with a reasonable computational time and is easy to implement (Klein, 2000). The task selection is important to predict the outcome expressed by the objective function. The priority rules can be classified into two types (Haupt, 1989): local and global. The distinction between the local and global rules is in term of the information status rules. In local rules, the information is about the waiting jobs at the machine for the current time. A global rule requires information about the jobs beyond the corresponding queue. The local information of the queue can be applied to our parallel processing system with centralized and no splitting structure. In local priority rule, the information is also more specific to the workstation. The local rules are widely used because they are robust against disruption (Hartmann et al., 2004). Our local rule has time independent priority computation. Obviously, the processing time does not change over time once they have been computed in a queue.

In the parallel processing system, ready tasks share a single waiting list. In multi-step method, First-Come-First-Serve (FCFS) priority rule is computed at the first stage of the method and produces a local priority rule (LPR) scheduling list, Q_{LPR}. Figure 3.3 depicts the the process of the LPR. FCFS is a random attribute in the local queue and a simple task priority rule which can often be found in real-systems but is usually considered inadequate by others. Note that, a FCFS discipline implies that services begin in the same order as arrival, but that tasks may have the system in a different order because of different-length service time. The multi-step scheduler serves the unscheduled tasks in the waiting list using a specific task se-

```
Procedure Local_Priority_Rule()
  begin
      for t = 1 to t=t_max do
          if t_1≤t_2≤...≤t_max then
              for i = 1 to n do
                  Q_list:=perform candidates for Q_LPR;
              endfor
          endif
      endfor
  end
```

Figure 3.3: Pseudo-algorithm for Local Priority Rule

lection process that will discuss in the next subsection.

In Baccelli et al. (1993), FCFS policies have been seen that are optimal in terms of the number of tasks and the throughput (i.e. the number of activities that run to completion within the given amount of time). There are cases where FCFS also exhibit a nice property regarding response times and smaller vectors of transient response times for all tasks. However, sometimes the optimality properties of FCFS fail to hold when tasks entering the system have such a random structure. The multi-step method can be effectively employed to improve the system. In order to escape from this randomization problem, we propose the following two modification procedure in the multi-step method loop: Cluster Insertion (CI) and Local Cluster Interchange (LCI).

Cluster Insertion (CI)

We define an insertion for the $P|on-line, r_i|C_{max}$ problem in the tasks sequence where one task is moved from a current Q_{list} to an improved Q_{list}. The CI procedure is a build up phase for refinement purpose in the local priority rule loop. In this phase, the tasks are extracted from Q_{LPR} and inserted one by one to a build up phase that constructs a cluster. We employ this CI idea from clustering method

in proposed heuristic algorithm in the literature for the general clustering problem. Most of them consider the clustering problem in off-line scheduling. More detail, clustering is a method of gathering tasks together and mapping them in the same group. Formally, the clustering method is either based on linear clustering or non-linear clustering. Linear clustering is a situation where the task gather in the same cluster and dependent to each other as they have precedence relations among them. Our CI process categories in non-linear clustering. The step involves two or more tasks in a cluster where the tasks are independent. These independent tasks can be easily distributed to parallel processors since there is no interference among the processors and other tasks. Therefore, there is no restriction in partitioning the tasks in CI process. CI can provide some advantages when the task is moved into several partitions. The random structure of the arrival tasks in the system could be simply managed. Therefore, the randomization issue in FCFS local priority rule can be partly solved by CI and completely solved in the LCI phase that we will discuss in the next section.

Obviously, in order for the movement to be accepted, firstly, the task i has to be in the Q_{LPR} i.e $J_i \in Q_{LPR}$. Secondly, the release date of the task must be greater than the arrival time. Otherwise the mapping won't succeed. Notice that this system is continuous and idle time between two tasks is unaccepted. More specifically, for the $P|on-line, r_i|C_{max}$ problem, the CI procedure consists of dividing tasks in Q_{LPR} and insert them into particular ζ slots. This insertion procedure are essentially repeated by n times without repetition. All mapped tasks are deleted from Q_{LPR} and inserted into a new list denoted as Q_{CI}. Figure 3.4 shows a clear description of this procedure of CI.

Local Cluster Interchange (LCI)

The LCI method is a permutation process specifically employed in the multi-step method loop after CI is adopted. In this process, the clusters that have been con-

```
Procedure Cluster_Insertion( )
    begin
        Candidates Q_LPR;
        while (status=1) do
            for i = 1 to n do
                for t = 1 to t = t_max do
                    a_{i,t_a} := the arrival of task i at time t_a;
                    r_{i,t_b} := the release date of task i at time t_b;
                    if (a_{i,t_a} < r_{i,t_b}) then
                        for k = 1 to k = ζ do
                            Extract candidate from Q_LPR and assign to cluster;
                            Q_list := perform candidates for Q_LI;
                        endfor
                    else
                        Processor_Selection ();
                    endif
                endfor
            endfor
        endwhile
    end
```

Figure 3.4: Pseudo-algorithm for Cluster Insertion procedure

structed in Q_{CI} will have mutation process to produce a new Q_{list} named Q_{LCI}. The mutation technique works by swapping the tasks within the local cluster in order to control the structure of the system. The task swapping process in cluster A will apply a simple and fast list scheduling (LS) algorithm to produce permutation cluster A'. This step is similarly applied until all clusters are done. To initiate the permutation process, the first cluster in Q_{CI} is chosen and stored as ζ_1. Then, the procedure continues for the second cluster in the list. The second cluster is stored as ζ_2. This step is then applied for all clusters in Q_{CI}. Once this stage is accomplished, the local cluster interchange operation begins.

For a Q_{LCI} to be obtained, only one LS algorithm can be applied for the whole Q_{list}. In this proposed heuristic, we opt two well known LS algorithms for the interchange process. More specifically, the LS for the problem $P|on-line, r_i|C_{max}$ are Longest Processing Time (LPT) and Shortest Processing Time (SPT) algorithms. For every cluster, we evaluate the processing time according to the selected LS method. Then, after the LS procedure, all the tasks in the cluster are ready in their own positions. If the task ahead is already assigned, the position is empty. The next task is able to replace and transfer to the empty position and the process continues for all remaining task in the Q_{LCI}.

In LPT algorithm, a starting point or starting tasks must be chosen for a cluster is by the longest processing time among the task i.e $p_1 \geq p_2 \geq \cdots \geq p_i$. The same procedure is repeated until the last cluster is reached. Similarly, the SPT also has to define the starting task and continue until the final task in a cluster but by using the shortest processing time approach where $p_1 \leq p_2 \leq \cdots \leq p_i$ is applied. This procedure is continue to form Q_{LCI}. The owner of the first position in Q_{LCI} is the first task which will be picked up for the next procedure. The pseudo-algorithm for LCI procedure is shown in Figure 3.5.

```
Procedure Local_Cluster_Interchange()
  begin
      Candidates Q_CI;
      while (status=1) do
          for ζ = 1 to ζ = ζ_max do
              store the cluster accordingly as ζ_1 , ζ_2 , ..., ζ_max ;
              for i = 1 to i = J_max do
                  swap the tasks and sort according to the LS algorithm;
                  Q_list:=perform candidates for Q_LCI;
              endfor
          endfor
      endwhile
  end
```

Figure 3.5: Pseudo-algorithm for Local Cluster Interchange procedure

3.2.2 Processor Selection Scheme

We know present a processor selection algorithm after multi-step loop in the task selection process. We have a list of candidates that are ready to get served. In on-line scheduling, the random processor selection is not the wisest approach. Therefore, the best way of choosing which processor to be assigned once a task has been selected is by the greedy way. We maintain our heuristics with a simple, fast and also can produce good results in this dynamic environment. The algorithm that we apply is a search method for the earliest processor which is idle the first to compose the selected task. The status of the processor will be updated for each time slot. During the tracking, if an available processor is found at the time slot, the status of the processor will be 0, otherwise 1. The selected task is accepted to be assign to the first processor with status 0 only. However, there might be more that one processor with status 0 at same time t. In this case, the task can choose the processor with the smaller index. In brief, the algorithm can be stated as follows:

Step 1: Define the status of the processors. If the processor is busy, the status is 1 otherwise it is 0 for idle.

Step 2: At time $t = 0$, initialize the status of all processors with 0. This means that, all processors are idle and there are no tasks in the system.

Step 3: At time $t \geq 0$, check the status of the processor at each time slot from the lowest index to the highest index label. Select the first processor with status 0 and assign to the task. Discard other processors with status 1.

Step 4: If there are no processor available at that time slot, discard all processors and continue checking the status for the next time slot. Repeat the process for all processor at each time slot until all task have been assigned.

The final step of the heuristic is the status update phase. The status of the system must be updated to reflect the new changes. The system will stop when all tasks are assigned to the processors.

3.2.3 A Small Example

We use a simple example to illustrate the procedure and flow of our proposed heuristic. The first step of the example starts from time t for the current state of the system. The following steps discuss in detail of the heuristic.

Step 1: Current state of the system.

Table 3.1 holds the information of the system at time t. There are five processors and running five current tasks which are J_4, J_5, J_6, J_7 and J_8. Since J_6 and J_4 are idle, a new task in the Q_{list} has to be allocated.

Step 2: Schedule a new task.

The current waiting tasks with their processing times is shown in Table 3.2. Notice that for this example, we choose LPT as the selected LS into LCI step. There are three remaining tasks left in Cluster 1. The number in brackets indicates the processing time of the task. At this stage, the first and second position (Pos 1 and Pos 2) in the Q_{list} are selected. Therefore, J_9 and J_{10} are assigned to the M_1 and M_4 respectively. The update status of the processor and the completion time for the new scheduled tasks are shown in Table 3.3.

Step 3: Incoming tasks.

At the same time t, a new set of incoming tasks arrive in the system and shown as follows: $\{J_{12}(3), J_{13}(5), J_{15}(1), J_{16}(2)\}$. Note that, in this step, it is not necessarily the system that always receive incoming task for every time slot. If there is no incoming task at a certain period, the system will directly continue to step 5.

Step 4: Task selection process.

The new set of arrival tasks are moved from the Q_{LPR} and we construct a new cluster in the CI phase to produce Q_{CI}. Then, the LCI approach is applied to form Q_{LCI}. The update waiting tasks are shown in Table 3.4.

Step 5: Continue for the next time slot.

At time $t+1$, M_3 and M_5 are available as shown in Table 3.5 and J_{11} and J_{13} are ready to get served on M_3 and M_5 as shown in Table 3.6. The new status and completion time are updated for $t+1$ as in Table 3.7. This procedure will repeat until all tasks in the Q_{LCI} are assigned and no task in the system.

Table 3.1: Information at time t

Processor	Current Task	Processor Status	Completion Time
M_1	J_6	0	t
M_2	J_5	1	$t+2$
M_3	J_8	1	$t+1$
M_4	J_4	0	t
M_5	J_7	1	$t+1$

Table 3.2: Q_{LCI} at time t

Cluster	ζ_1		
Task	J_9(4)	J_{10}(2)	J_{11}(1)
Position	Pos 1	Pos 2	Pos 3

Table 3.3: Update information at time t

Processor	Current Task	Processor Status	Completion Time
M_1	J_9	1	$t+4$
M_2	J_5	1	$t+2$
M_3	J_8	1	$t+1$
M_4	J_{10}	1	$t+2$
M_5	J_7	1	$t+1$

Table 3.4: Update Q_{LCI} at time t

Cluster	ζ_1	ζ_2			
Task	J_{11}	J_{13}	J_{12}	J_{16}	J_{15}
Position	Pos 1	Pos 2	Pos 3	Pos 4	Pos 5

Table 3.5: Information at time $t+1$

Processor	Current Task	Processor Status	Completion Time
M_1	J_9	1	$t+4$
M_2	J_5	1	$t+2$
M_3	J_8	0	$t+1$
M_4	J_{10}	1	$t+2$
M_5	J_7	0	$t+1$

Table 3.6: Update information at time $t+1$

Processor	Current Task	Processor Status	Completion Time
M_1	J_9	1	$t+4$
M_2	J_5	1	$t+2$
M_3	J_{11}	1	$t+2$
M_4	J_{10}	1	$t+2$
M_5	J_{13}	1	$t+6$

Table 3.7: Update Q_{LCI} at time $t+1$

Cluster	ζ_2		
Task	J_{12}	J_{16}	J_{15}
Position	Pos 1	Pos 2	Pos 3

3.3 Computational Experiments

In this section we present a simulation experiment on the proposed algorithm for solving the $P|on-line, r_i|C_{max}$. The implementation is performed to evaluate the effectiveness of the proposed algorithm. For this purpose, we present three implementations of the algorithms and test their performances. We evaluate the comparison among these heuristics and present the best heuristic. We also reveal the gaps with the optimum value. As we know the proposed algorithms are on-line scheduling and there are no optimum algorithm is obtain for the schedule. Therefore, we perform off-line scheduling to compute the optimum result even we know that it is not comparable in terms of getting the task information. The comparison is to measure the performance of our algorithm as in undeterministic condition compare to deterministic condition. We hope that our algorithms have as small as possible gap of the objective function between the optimal off-line schedule. The following are the three heuristics that we use in the simulation testing:

HA 1: The HA 1 algorithm implements LPR, CI and LCI in the task selection process. In LCI, each of the $\zeta_1, \zeta_2, \ldots, \zeta_{max}$ contains a random list scheduling with no specific sequence order to form Q_{LCI}.

HA 2: This algorithm puts together the multi-step method in the overall algorithm in Figure 3.2 with LPR, CI and LCI for the task selection phase. It has to be noted that in LCI, HA 2 has applied LPT as the formation in the interchange procedure. The final Q_{list} obtain by HA 2 is Q_{LCI} before they get transferred to processor selection stage.

HA 3: As HA1 and HA 2, HA 3 also carry out the procedure in this order: LPR, CI and LCI. The final Q_{list} of the the task selection multi-step in HA 3 is Q_{LCI}. We also have to remark that in LCI of HA 3 there are slightly modified from HA 2 in permutation process. We obtain SPT as the LS to implement the interchange before agreed to the next level which is processor selection.

3.3.1 Experimental Design

We implement our HA 1, HA 2 and HA 3 using Microsoft Visual C++ 6.0 on a personal computer with Intel Core 2 2.66 GHz 1.95 GB RAM. The algorithms are in a dynamic environment where the task information can only be known during the execution over time. We generate optimum solution for the problem using AIMMS 3.10 software package. The simulation data for the problem $P|on-line, r_i|C_{max}$ is generated as follows:

1. The number of independent tasks are $n = \{200, 400, 600, 800, 1000\}$.
2. For every set of tasks, we have $\zeta = \{10, 20, 30, 40, 50\}$.
3. For every combination of n and ζ, we have $m = 3$ and $m = 5$.
4. The processing time for the instances is assumed to follow a discrete uniform distribution between 1 to 60 i.e. distribution $U[1, 60]$.

5. We generate 20 instances for every combination. Therefore, in this experiment, we have $n \times \zeta \times m = 5 \times 5 \times 2 = 50$ combinations that produced $50 \times 20 = 1000$ instances.

3.3.2 Computational Results

We now present our performance results of the HA 1, HA 2 and HA 3 algorithms compare with the optimal solutions. We report the solutions for the objective function (i.e. makespan) for every instance, I. The average percentage deviation from the optimum, Gap_a, are examined and can be calculated as follows:

$$Gap_a(\%) = \frac{1}{20}\sum_{I=1}^{20}\frac{C_{max}(I) - {C_{max}}^*(I)}{{C_{max}}^*(I)} \times 100 \tag{3.6}$$

where $C_{max}(I)$ is the makespan obtained by the heuristic for instance I and $C^*_{max}(I)$ is the makespan by the optimum solution of instance I.

The maximum gap, Gap_w, of the instances for every combination also have been observed by the following formulation:

$$Gap_w(\%) = \max\left\{\frac{C_{max}(I) - {C_{max}}^*(I)}{{C_{max}}^*(I)} \times 100 \;\middle|\; I = 1, 2, \ldots, 20\right\} \tag{3.7}$$

In the following, we present the computational results obtained from the simulation of HA 1, HA 2 and HA 3 algorithms. The results will be discussed according three different aspects. In the first part, we discuss the results delivered by the best heuristics. Then, we concentrate on the impact of the number of cluster to the quality of solution obtained by HA2 and HA3. Lastly, we analyze the performance of the algorithms for the case when the tasks in the system get larger.

Performances of the best heuristics In this testing, we have carried out a large data set for the experiment. Tables 3.8 and 3.9 give for each heuristic algorithms the average value for the gap compared with the optimum solutions as a function of the problem size (m, ζ, n). We also evaluate the algorithms with each other by reporting the number of instances in which the heuristic becomes the best heuristic denoted as NBH. If there more than one heuristic obtained the same best solution for a certain instance, those particular algorithms can be declared as the best heuristic for that instance. We also reported the Gap_w, for every combination from the 20 instances to observe the worse performance by the algorithms. It is observed that all instances in the problem shown in tables 3.8 and 3.9 can be solved within 1 second on average.

Number of Processor m	Number of Cluster ζ	Number of Tasks n	HA1			HA2			HA3		
			$Gap_a(\%)$	$Gap_w(\%)$	NBH	$Gap_a(\%)$	$Gap_w(\%)$	NBH	$Gap_a(\%)$	$Gap_w(\%)$	NBH
3	10	200	0.922	1.642	1	0.119	0.296	20	3.405	4.014	0
		400	0.464	0.869	0	0.050	0.099	20	1.831	2.030	0
		600	0.248	0.392	0	0.027	0.049	20	1.244	1.337	0
		800	0.180	0.396	0	0.018	0.039	20	0.982	1.062	0
		1000	0.152	0.320	0	0.013	0.029	20	0.769	0.832	0
	20	200	0.656	1.758	6	0.292	0.869	16	2.917	3.656	0
		400	0.337	0.680	1	0.060	0.126	20	1.687	1.885	0
		600	0.261	0.538	1	0.042	0.081	19	1.189	1.288	0
		800	0.222	0.446	1	0.022	0.050	20	0.917	1.023	0
		1000	0.165	0.247	0	0.014	0.030	20	0.748	0.819	0
	30	200	0.651	1.315	6	0.370	0.975	16	2.212	3.503	0
		400	0.385	0.954	2	0.104	0.330	19	1.503	1.883	0
		600	0.267	0.527	0	0.040	0.083	20	1.111	1.335	0
		800	0.237	0.411	0	0.026	0.063	20	0.911	0.983	0
		1000	0.156	0.322	1	0.022	0.050	20	0.719	0.840	0
	40	200	0.788	1.360	4	0.479	1.189	16	1.794	3.351	0
		400	0.384	0.697	4	0.179	0.697	17	1.378	1.659	0
		600	0.282	0.636	2	0.060	0.120	19	1.055	1.269	0
		800	0.186	0.397	1	0.034	0.114	19	0.817	0.956	0
		1000	0.146	0.305	1	0.025	0.059	19	0.681	0.766	0
	50	200	0.875	1.640	4	0.597	1.341	16	1.825	3.363	0
		400	0.357	0.929	2	0.117	0.347	17	1.194	1.705	0
		600	0.212	0.438	2	0.063	0.233	18	1.023	1.344	0
		800	0.193	0.398	0	0.036	0.088	20	0.874	0.984	0
		1000	0.162	0.373	2	0.019	0.040	19	0.670	0.748	0

Table 3.8: Comparison performance of the heuristic algorithm with the optimum solution for $m = 3$

Number of Processor m	Number of Cluster ζ	Number of Tasks n	HA1			HA2			HA3		
			$Gap_a(\%)$	$Gap_w(\%)$	NBH	$Gap_a(\%)$	$Gap_w(\%)$	NBH	$Gap_a(\%)$	$Gap_w(\%)$	NBH
5	10	200	2.091	10.398	1	0.892	9.185	19	6.130	14.644	0
		400	0.878	1.373	0	0.107	0.292	20	3.065	3.354	0
		600	0.519	0.895	0	0.063	0.194	20	2.123	2.261	0
		800	0.441	0.895	0	0.037	0.105	20	1.643	1.813	0
		1000	0.429	1.728	1	0.099	1.529	19	0.591	2.825	0
	20	200	1.758	2.914	0	0.715	1.529	20	4.277	5.578	0
		400	0.985	1.689	0	0.194	0.444	20	2.651	3.224	0
		600	0.666	1.125	0	0.098	0.195	20	1.985	2.363	0
		800	0.380	0.631	1	0.042	0.100	20	1.537	1.774	0
		1000	0.273	0.558	1	0.040	0.082	20	1.256	1.336	0
	30	200	1.681	3.339	4	0.992	2.323	16	2.821	4.672	0
		400	0.930	0.930	1	0.206	0.759	20	2.448	3.133	0
		600	0.598	0.916	0	0.148	0.283	20	1.803	2.104	0
		800	0.455	0.747	1	0.073	0.165	20	1.418	1.646	0
		1000	0.274	0.589	0	0.045	0.100	20	1.132	1.336	0
	40	200	1.696	3.247	1	0.942	2.248	19	2.690	3.620	0
		400	0.904	1.376	1	0.425	1.262	19	2.122	2.851	0
		600	0.675	1.084	0	0.145	0.352	20	1.688	2.051	0
		800	0.398	0.772	1	0.097	0.298	20	1.305	1.693	0
		1000	0.322	0.694	0	0.061	0.181	20	1.137	1.308	0
	50	200	1.647	2.758	4	1.099	2.473	18	2.653	3.986	0
		400	0.709	1.319	7	0.520	1.401	14	1.642	2.613	0
		600	0.547	0.898	0	0.204	0.515	20	1.446	1.967	0
		800	0.414	0.804	1	0.094	0.210	20	1.322	1.711	0
		1000	0.358	0.587	0	0.060	0.149	20	1.107	1.350	0

Table 3.9: Comparison performance of the heuristic algorithm with the optimum solution for $m = 5$

The tables already give us clear observation that the algorithms of HA 1, HA 2 and HA 3 achieved a very good performance, where the average gap is less than 6.13% from the optimum for all size combinations. The best heuristic is HA 2, which is very outstanding, when the maximum Gap_a is only 1.099% for the case $m = 5, \zeta = 50$ and $n = 200$. As can clearly be seen from the data in the table, HA2 always obtains the lowest Gap_a for all different size of problems. From the NBH, HA 2 has be the best heuristic with percentage of 95.4 from 1000 generated test problems. The HA 1 and HA 3 also produced efficient results with 2% and 6% for the same case $m = 5, \zeta = 10$ and $n = 200$ of the maximum Gap_a respectively.

Performances of the heuristics with increasing number of clusters Figures 3.6 and 3.7 show the average Gap_a of different number of clusters for $m = 3$ and $m = 5$ respectively. In the diagrams, the x-axis indicate the number of clusters and the y-axis is the average of Gap_a for $n = 200$ to $n = 1000$. The graph illustrates the performance of the heuristics as the number of clusters increase while the number of tasks stay fixed. It can be seen from the figures that, an interesting observation from the least heuristic HA3, the average Gap_a have decreased as the number of cluster increase. The results show us the number of cluster influence the efficiency of the performance for HA3. The progress of HA3 getting better result when the cluster contain smaller amount of the arrival tasks.

For HA 1, the performance of the average Gap_a almost the same for all cluster as the random list is applied in the local cluster interchange procedure. However, for HA 2, which is the best heuristics, has unexpected performance when the algorithm has a slight increase as the cluster is growing in the system. But the behavior is still good because the most maximum average Gap_a is still very small which is 0.4%. Moreover, if we take a closer look for an example of $\zeta = 10$ and $m = 3$ of HA2 in the table, the Gap_a reduced from 0.1% for $n = 200$ to 0.01% for $n = 1000$. This reduced pattern is happen for all combination of clusters and processors. If we fixed

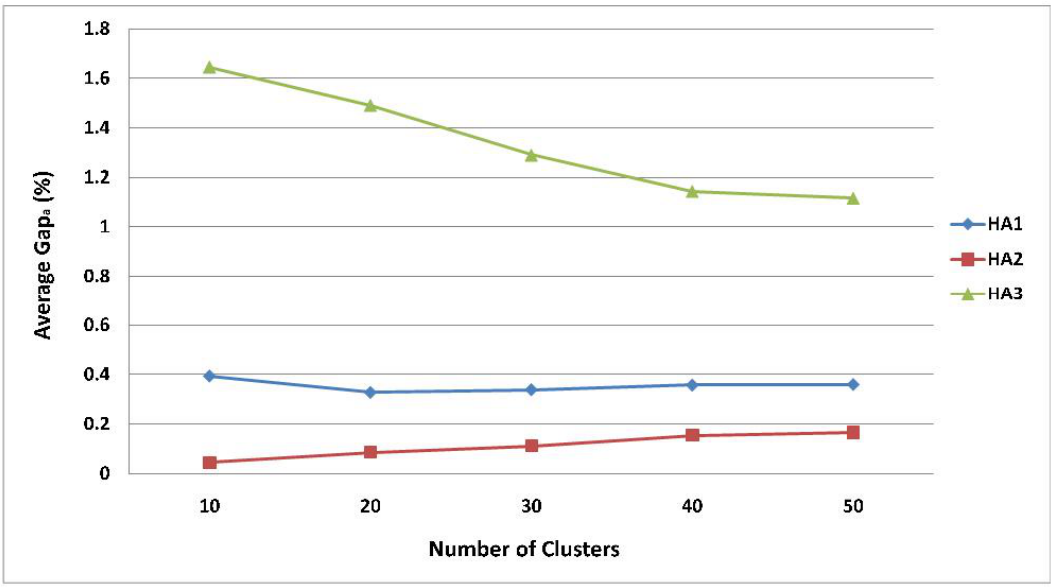

Figure 3.6: Comparative performance of HA 1, HA 2 and HA 3 on $m = 3$ for $\zeta =$ 10 to 50 .

the number of cluster, the HA2 perform well even the size of the system is increases.

Performances of the heuristics with increasing number of tasks Figures 3.8 and 3.9 show the average Gap_a for HA 1, HA 2 and HA 3. From the figures, we want to show the performance of the heuristics when the instances get larger in the system. Normally, as the size of the problem in a system increases, the performance of algorithms becomes unstable and produce less impressive results. However, in this experiment, all the three algorithms obtain a very surprisingly good performance when the value of the average Gap_a are decreasing as the number of the tasks in the system are increasing.

In terms of the best performance with the largest percentage in reducing the average Gap_a HA 3 done that. This is followed by HA 2 and HA 1. For $m = 3$, we observed that HA 3 reduced the average Gap_a from 2.4% for $n = 200$ to 0.7% for $n = 1000$. As $m = 5$, we also see the largest drop of HA 3 among other algorithms with average Gap_a difference 2.7% from $n = 200$ to $n = 1000$. The HA 1 and

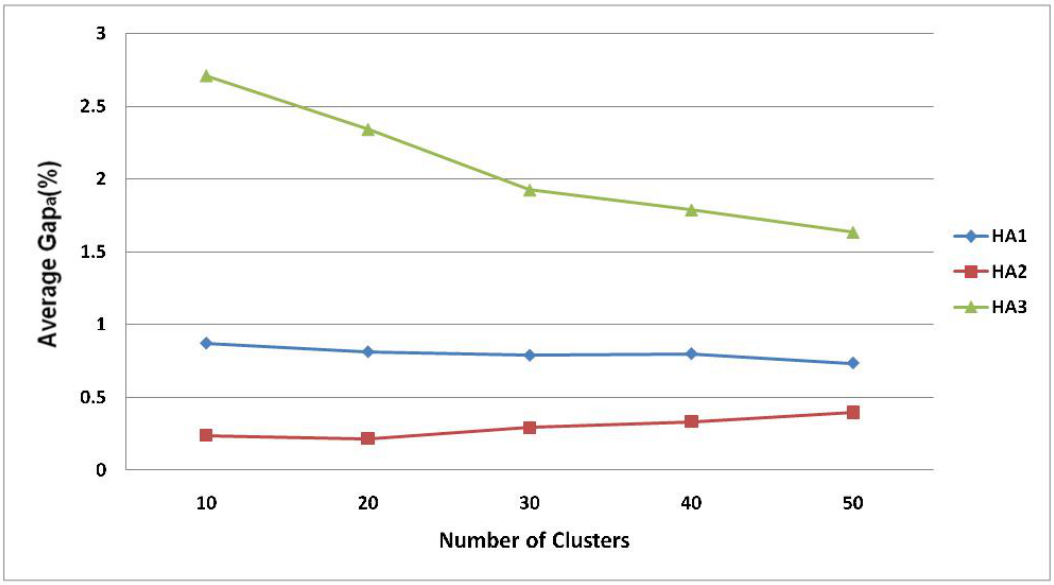

Figure 3.7: Comparative performance of HA 1, HA 2 and HA 3 on $m = 5$ for $\zeta =$ 10 to 50 .

HA 2 results for the average Gap_a are already the best two algorithms in term of the average Gap_a performance. As in the diagram, all the three heuristics show the quality performance with nearly getting to optimum as the system size increases. For the case of $n = 1000$ and $m = 3$, the problem recorded the average gap of 0.15%, 0.02% and 0.72% for HA 1, HA 2 and HA 3 respectively. For the case of $n = 1000$ and $m = 5$, the average gap are approximately 0.33% for HA 1, 0.06% for HA 2 and 1.04% for HA 3. Considering the improvement in the quality of performance, all the three algorithms are very efficient in solving $P|on-line, r_i|C_{max}$.

3.4 Summary

In this chapter, we introduced one of the additional features in parallel processing system that we explored in our studies which is on-line scheduling with release date in minimizing the makespan. This feature is dynamic and therefore, we applied a multi-step method to reduce the non-determinism in the on-line scheduling. We partition the scheduling process into three phases:(1) local priority rule; (2) cluster

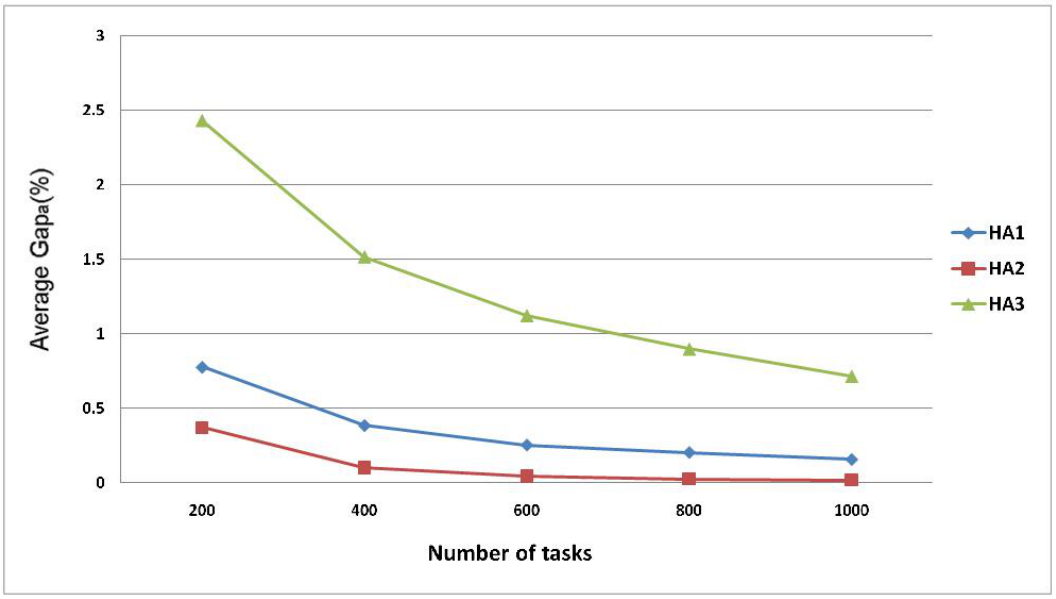

Figure 3.8: Comparative performance of HA 1, HA 2 and HA 3 for $n = 200$ to 1000 on $m = 3$.

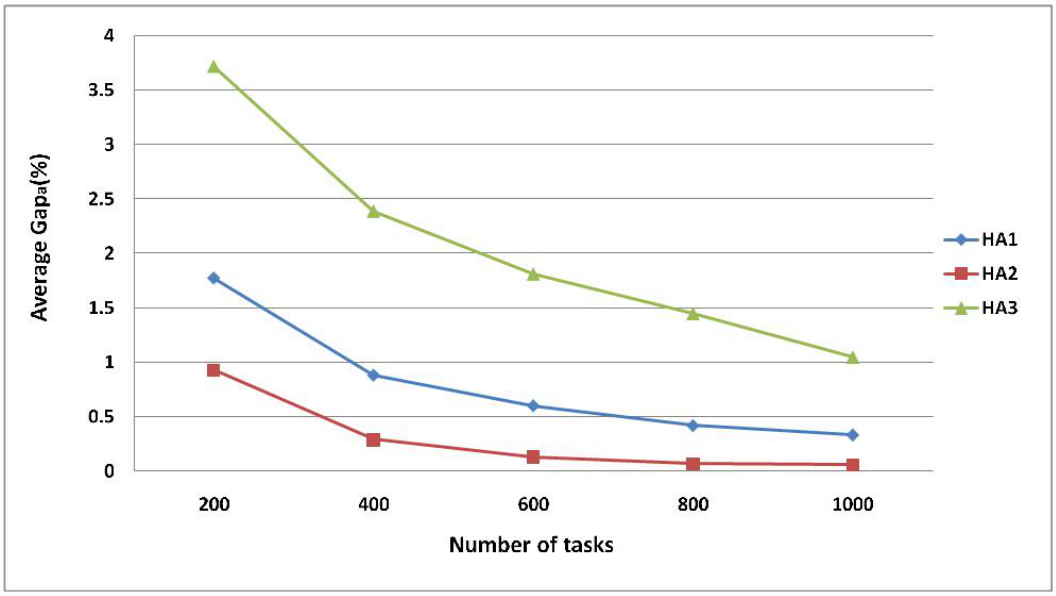

Figure 3.9: Comparative performance of HA 1, HA 2 and HA 3 for $n = 200$ to 1000 on $m = 5$.

insertion; and (3) local cluster interchange. The different phases in the multi-step is the improvement for the next step in the algorithm. We are interested in these algorithms by their simplicity and practical usage in real practices. All the three algorithms are very efficient and produced a very good quality result with the average gap nearly optimum. The result is very impressive since all the information are not known before hand and this problem categories in NP-hard optimization problem. HA 2 is reported as the best heuristic with the smallest average gap and always be the best heuristics. All the heuristics performed very well when the algorithms are able to improve the results when the size of the system increases.

Chapter 4

Priority Consideration in Parallel Processor System

The previous chapter considered the on-line scheduling problem. In this chapter we explore off-line scheduling with another additional feature, that relates to task priority, in the parallel processing system. Note that, the priority consideration is different to precedence constraint. In precedence constraint, job a is restricted to start processing before job b is finished. While in priority consideration, job a may get priority to start and get served even if job b is still not complete the processing. In addition, we consider the priority in unrelated parallel processor environment that has many applications in industry. A mixed integer linear programming (MILP) model is developed to obtain optimal solution for the problem. We present several models for different types of ordering for the system.

This chapter is organized as follows. In Section 4.1, the specific descriptions and assumptions are discussed. Section 4.2 presents the developed mathematical model for priority restrictive requirement. In Section 4.3, we validate and implement the MILP model. Section 4.4 gives the summary of the results.

4.1 Problem Descriptions

The priority consideration involved in the parallel processor system has responsibility to the relation of an independent task (or probably more) that may require starting the processing earlier before processing begins on another processor. In contrast, a precedence relation between two tasks requires that a task cannot start before another task has been completed because the tasks are dependent. For example, in computer production factory where the motherboard has to be ready before the installation process of the software applications.

In our system, priority consideration is defined by an ordered list, L, that contain a set of independent tasks. Suppose task 1 is the first arrival task in the system followed by task 2. Then, task 2 cannot start processing before task 1 on the same processor, but task 2 can start processing if there is another available parallel processor that can accept the task for processing. Therefore, the rules where task 2 cannot start the processing before task 1 finish is only applicable on common processors. This type of situation can be found in real applications such as in the area that involve queuing systems in manufacturing and service industries. In addition, another application related to the priority consideration is flow scheduling in network applications such as on multihomed mobile hosts (Zafeiris and Giakoumakis, 2008). Zafeiris and Giakoumakis (2008) considered a problem of assigning traffic flows to available active radio interfaces and bearer services in the network selection to obtain the best available Radio Access Network (RAN) to the user. They modeled the problem of network selection and flow assignment in the context of a multihomed Multimode Mobile Terminal where the priority is based on the flow mobility supported by the network infrastructure. In other applications, priority of the task sequence may be important when dealing with the task with certain timing behavior of real-time systems (Andersson, 2003).

In order to illustrate further about the priority consideration between tasks,we

Counter	Visa	Enrolment
Counter 1	10 min	10 min
Counter 2	30 min	10 min

Table 4.1: Time taken for the visa and enrolment application

describe the concept by a simple example in a student central office in a university. The student central office is a place for the university to coordinate the services for all students. Suppose each counter in the student central serves all official type of services including enrolment for all students and visa application for international student. The arrangement in the student central allow the earlier student to get served first according to the order. Consider the situation where there are two counters open for business as in Table 4.1. Normally, the time taken to process the visa and enrolment process will be the same as for Counter 1. However, sometimes there is a technical problem that forces the service for visa application can take longer due to a manual process instead of on-line process as occurs in Counter 2. Suppose that three students have arrived in sequence to get served in the student central office. The students come to settle about the following services: visa only; visa and enrolment ; and enrolment only respectively. Figure 4.1 depicts two feasible situations (a and b) for priority consideration.

In the situation a as in Figure 4.1, counter 1 serves the student 1 (visa) and student 2 (enrolment and visa). Student 2 well have to wait until student 1 is finished the visa application. The priority consideration is given to the third student where the student is allowed to get serve for the enrolment from counter 2 earlier than the second student with enrolment and visa application. For situation b, counter 1 is attended by student 1 and student 3. Student 3 will be fully served after student 1 is finished. Student 2 get processed in sequence before student 3 but student 3 is allowed to get done before student 2.

The counters in the student central represent the unrelated parallel processors

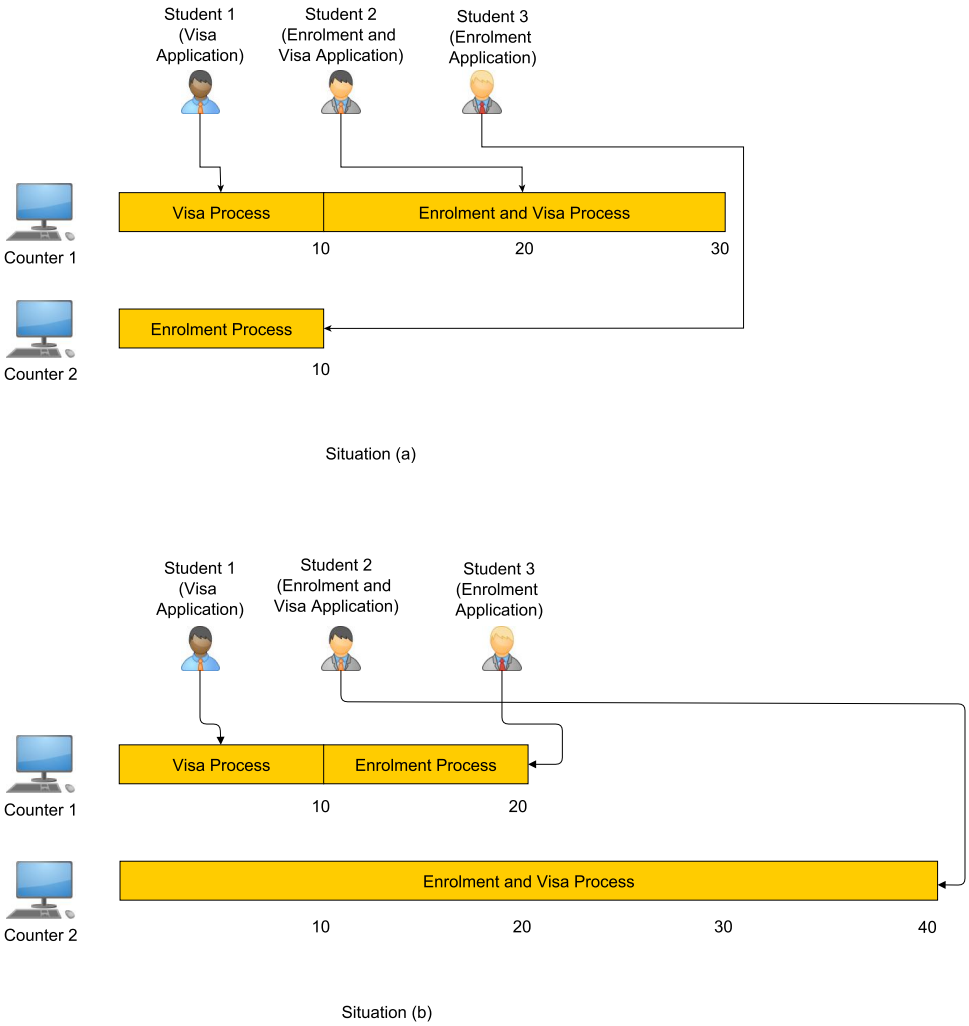

Figure 4.1: Feasible solutions for priority consideration

where the processing time for a same task is different on each processor. In this example, it is clear that the priority consideration is relevant for unrelated parallel processor system instead of identical processor. The unrelated parallel processor system with objective of minimizing the makespan (i.e. completion time of the last tasks for every processor) add the difficulty to the system and may also increase the computational complexity. Moreover, the unrelated parallel processor problem also has a structure that can be applied in real-time environment. (see Section 2.1.3).

Formally, the unrelated parallel processing system considered in this chapter has n independent jobs , J_i $(i = 1, 2, \ldots, n)$, and m unrelated parallel processors , M_j $(j = 1, 2, \ldots, m)$, that have different execution times p_{ij} of job i on machine j. Each processor is assumed to be continuously available and there is no idle time between the execution of a pair of jobs. There is no relationship between the machine speeds and each machine is capable of processing only one job at a time. Each job can be assigned to any one of the machines but job migration between processors is prohibited. All jobs are available for processing starting at time zero and there are no precedence constraints among them. The tasks are non-preemtive. Meaning that, each task needs to be processed by the same machine without interruption once it is started until completion. Since the processors are unrelated, the processing time of the tasks are dependent on the processors and this may differ for every processor even for the same task. In terms of complexity of deterministic scheduling, unrelated parallel processors scheduling problems are the most difficult to solve compared to identical and uniform parallel processors (Yu et al., 2002). Therefore, task scheduling problem with priority consideration in unrelated parallel processor systems is attractive to explore further.

The ordered list specifies the assignment of the tasks to the processor must be in sequence. Whichever task has been selected to be assigned to the specific processor according to the priority order will be processed as long as the processor is available. The ordered list of tasks is constructed by prioritizing the tasks

based on ascending order, descending order and a general priority list. In particular, we declare three sets of task list based on the priority and we denote as L_{ascend}, $L_{descend}$ and $L_{general}$. Our goal is to find an optimum schedule of n tasks that achieves all the constraints and the objective function. We consider minimizing makespan as the objective function where $y = C_{max} = max(C_j|j = 1, 2, \ldots, m)$. We specifically denoted the problem as $R|priority(ascending)|C_{max}$ for ascending order, $R|priority(descending)|C_{max}$ for descending order and lastly $R|priority(general)|C_{max}$ for general order.

In the next section, MILP models are developed to formulate the relation between the tasks involving priority consideration in unrelated parallel processors and optimizing the objective function, $R|priority|C_{max}$. This work has been published in Caccetta and Nordin (2010).

4.2 Mixed Integer Linear Programming Model for Priority Consideration

In this section, we develop a model for priority consideration to the problem $R||C_{max}$. Therefore firstly, we present the mixed integer programming model (MILP) of the $R||C_{max}$ problem that has been formulated by Potts (1985). We then extend the model and produce a formulation of $R|priority|C_{max}$ problem. The MILP model for the problem $R||C_{max}$ can be written as follows:

Let

$$x_{ij} = \begin{cases} 1, & \text{if task } i \text{ is assigned to machine } j \\ 0, & \text{otherwise.} \end{cases} \tag{4.1}$$

where $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, m$ and x_{ij} is an $0 - 1$ assignment variables.

$Minimize \quad y$

$s.t$

$$\sum_{j=1}^{m} x_{ij} = 1 \qquad \text{for } i = 1, 2, \ldots, n \tag{4.2}$$

$$\sum_{j=1}^{m} p_{ij} x_{ij} \leq y \qquad \text{for } i = 1, 2, \ldots, n \tag{4.3}$$

$$y \geq 0 \tag{4.4}$$

$$x_{ij} \in \{0, 1\} \qquad \text{for } i = 1, 2, \ldots, n \qquad j = 1, 2, \ldots, m \tag{4.5}$$

Constraints (4.2) ensure that each task is assigned to only one of the m machines. Constraints (4.3) ensure that the sum of execution time for every task on machine j is less than or equal to C_{max}.

For the purpose of defining the value of the total execution time for each machine j, additional constraints to the model (4.2) $-$ (4.5) can be written in the following form:

$$y_j = \sum_{i=1}^{n} p_{ij} x_{ij} \qquad \text{for } j = 1, 2, \ldots, m \tag{4.6}$$

$$y_j \leq y \qquad \text{for } j = 1, 2, \ldots, m \tag{4.7}$$

4.2.1 Priority in Ascending Order

Priority in ascending order is a list of task where the order of the tasks is in upward ranked where $L_{ascend} = \{L_{a_1}, L_{a_2}, \ldots, L_{a_n}\} = \{J_{i_1}, J_{i_2}, \ldots, J_{i_n}\}$ and $i_1 < i_2 < \cdots < i_n$. Scheduling an unrelated machine with priority in ascending order $R|priority(ascending)|C_{max}$ can be addressed by the following job sequencing assignment variables:

$$z_{ikj} = \begin{cases} 1, & \text{if processing of} J_i \text{ precedes processing of } J_k \text{ on } M_j \\ 0, & \text{otherwise.} \end{cases} \tag{4.8}$$

and

$$z_{0kj} = \begin{cases} 1, & \text{if processing of} J_k \text{ is the first task on } M_j \\ 0, & \text{otherwise.} \end{cases} \tag{4.9}$$

The MILP formulation is as follows:

$Minimize \qquad y$

$s.t$

Constraints (4.2) $-$ (4.5) and

$$\sum_{k=1}^{n} z_{0kj} = 1 \qquad \text{for } j = 1, 2, \ldots, m \tag{4.10}$$

$$\sum_{i \in I_k} \sum_{j=1}^{m} z_{ikj} = 1 \qquad \text{for } k = 1, 2, \ldots, n \tag{4.11}$$

$$\sum_{k \in K_i} \sum_{j=1}^{m} z_{ikj} \leq 1 \qquad \text{for } i = 1, 2, \ldots, n \tag{4.12}$$

$$\sum_{i \in I_k} \sum_{j=1}^{m} p_{kj} z_{ikj} \leq y \qquad \text{for } k = 1, 2, \ldots, n \tag{4.13}$$

$$(p_{ij} + p_{kj}) z_{ikj} \leq p_{ij} x_{ij} + p_{kj} x_{kj} \qquad i \in I_k \tag{4.14}$$

$$\text{for} \quad i = 1, 2, \ldots, n \qquad k = 1, 2, \ldots, n \qquad j = 1, 2, \ldots, m$$

$$z_{ikj} \in \{0, 1\} \qquad \text{for } i = 1, 2, \ldots, n \qquad k = 1, 2, \ldots, n \qquad j = 1, 2, \ldots, m \tag{4.15}$$

Constraints (4.10) ensure that only one processing job will start first on each machine. Constraints (4.11) ensure that each job with ascending order should be processed by at least one machine. Constraints (4.12) guarantee the assignment of at most one job to each position on each machine. Constraints (4.13) ensure that the total execution time for every machine is less than or equal to the makespan. Finally, constraints (4.14) are to make sure that if $z_{ikj} = 1$ then J_i precedes the processing of J_k on the same M_j.

In constraints (4.11), (4.13) and (4.14), I_k represent the priority condition for J_i that must precede J_k on M_j. Notice that these constraints must meet the condition where $i < k$ and $i \neq k$ for $i = 0, 1, 2, \ldots, n$, for the task process on M_j. In general, this translates to the following:

$$\begin{aligned}
&\text{for} \quad k = 1, I_1 = \{i_0\} \\
&\text{for} \quad k = 2, I_2 = \{i_0, i_1\} \\
&\text{for} \quad k = 3, I_3 = \{i_0, i_1, i_2\} \\
&\text{for} \quad k = n, I_n = \{i_0, i_1, i_2, \ldots, i_{n-1}\}
\end{aligned}$$

In constraints (4.12), K_i refer to the sequence assignment condition for J_i that must be followed by J_k where $i < k$ and $i \neq k$ for $k = 1, 2, \ldots, n$ when assigned to M_j. The following gives the detail of this condition:

$$\begin{aligned}
&\text{for} \quad i = 1, K_1 = \{i_{i+1}, i_{i+2}, \ldots, i_{n-1}, i_n\} \\
&\text{for} \quad i = n-2, K_{n-2} = \{i_{n-1}, i_n\} \\
&\text{for} \quad i = n-1, K_{n-1} = \{i_n\} \\
&\text{for} \quad i = n, K_n = \{\}
\end{aligned}$$

4.2.2 Priority in Descending Order

Descending order priority is an ordered list of downward rank of tasks which is $L_{descend} = \{L_{d_1}, L_{d_2}, \ldots, L_{d_n}\} = \{J_{i_1}, J_{i_2}, \ldots, J_{i_n}\}$ and $i_1 > i_2 > \cdots > i_n$. The scheduling model of unrelated parallel machines with priority in descending order, $R|priority(descending)|C_{max}$, can be addressed by variables (4.8) and the following $0-1$ job sequencing assignment variable:

$$z_{i0j} = \begin{cases} 1, & \text{if processing of task } i \text{ is the last task on machine } j \\ 0, & \text{otherwise.} \end{cases} \tag{4.16}$$

The MILP formulation is as follows:

$Minimize \quad y$

$s.t$

Constraints (4.2) $-$ (4.5) and

$$\sum_{i=1}^{n} z_{i0j} = 1 \qquad \text{for } j = 1, 2, \ldots, m \tag{4.17}$$

$$\sum_{k \in K_i} \sum_{j=1}^{m} z_{ikj} = 1 \qquad \text{for } i = 1, 2, \ldots, n \tag{4.18}$$

$$\sum_{i \in I_k} \sum_{j=1}^{m} z_{ikj} \leq 1 \qquad \text{for } k = 1, 2, \ldots, n \tag{4.19}$$

$$\sum_{k \in K_i} \sum_{j=1}^{m} p_{ij} z_{ikj} \leq y \qquad \text{for } i = 1, 2, \ldots, n \tag{4.20}$$

$$(p_{ij} + p_{kj}) z_{ikj} \leq p_{ij} x_{ij} + p_{kj} x_{kj} \qquad i \in K_i \tag{4.21}$$

$$\text{for} \quad i = 1, 2, \ldots, n \qquad k = 1, 2, \ldots, n \qquad j = 1, 2, \ldots, m$$

$$z_{ikj} \in \{0,1\} \qquad \text{for } i = 1,2,\ldots,n \qquad k = 1,2,\ldots,n \qquad j = 1,2,\ldots,m \tag{4.22}$$

Constraints (4.17) ensure that only one processing job will be the last on each machine. Constraints (4.18) ensure that each job with descending order should be processed by at least one machine. Constraints (4.19) guarantee the assignment of at most one job to each position on each machine. Finally, constraints (4.20) ensure that the total execution time for every machine is less than or equal to the makespan and constraints (4.21) are for tasks J_i and J_k to be processed on the same machine M_j if $z_{ikj} = 1$.

Here, we perform the details of the priority condition in constraints (4.18), (4.20) and (4.21) that must be fulfilled by set K_i when the task runs on M_j. These conditions satisfy the tasks where $i \neq k$ and $i > k$ for $k = 0,1,2,\ldots,n$.

$$\begin{aligned}
&\text{for} \quad i = 1, K_1 = \{i_0\} \\
&\text{for} \quad i = 2, K_2 = \{i_0, i_n, i_{n-1}\} \\
&\text{for} \quad i = 3, K_3 = \{i_0, i_n, i_{n-1}, i_{n-2}\} \\
&\text{for} \quad i = n, K_n = \{i_0, i_n, i_{n-1}, \ldots, i_{n-(n-2)}\}
\end{aligned}$$

For constraints (4.19), set I_k must also obey the assignment condition where $i \neq k$ and $i > k$ but for $i = 1,2,\ldots,n$.

$$\text{for} \quad k = 1, I_1 = \{i_1, i_2, \ldots, i_{n-1}\}$$

$$\text{for} \quad k = 2, I_2 = \{i_1, i_2, \ldots, i_{n-2}\}$$

$$\text{for} \quad k = n-2, I_{n-2} = \{i_1, i_2\}$$

$$\text{for} \quad k = n-1, I_{n-1} = \{i_1\}$$

$$\text{for} \quad k = n, I_n = \{\}$$

4.2.3 Priority with General Priority List

The task scheduling on unrelated parallel processors with general priority list denoted as $R|priority(general)|C_{max}$. The model is a generalization for all types of task ordering that has been described above where the priority list can be stated as follows: $L_{general} = \{L_{g_1}, L_{g_2}, \ldots, L_{g_n}\} = \{J_{i_1}, J_{i_2}, \ldots, J_{i_n}\}$ and $i_1, i_2, \ldots, i_n$ is a list of tasks according to a certain priority requirement. Which ever tasks need to be done earlier will be at the front of the list. The assignment variables (4.8) and (4.16) are used in the model and the scheduling $R|prioritylist|C_{max}$ can be formulated as follows:

$Minimize \qquad y$

$s.t$

Constraints (4.2) $-$ (4.5) and

$$\sum_{i=1}^{n} z_{i0j} = 1 \qquad \text{for } j = 1, 2, \ldots, m \tag{4.23}$$

$$\sum_{k \in K_{i_n}} \sum_{j=1}^{m} z_{ikj} = 1 \qquad \text{for } i = 1, 2, \ldots, n \tag{4.24}$$

$$\sum_{i \in I_{i_n}} \sum_{j=1}^{m} z_{ikj} \leq 1 \qquad \text{for } k = 1, 2, \ldots, n \tag{4.25}$$

$$\sum_{k \in K_{i_n}} \sum_{j=1}^{m} p_{ij} z_{ikj} \leq y \qquad \text{for } i = 1, 2, \ldots, n \tag{4.26}$$

$$(p_{ij} + p_{kj}) z_{ikj} \leq p_{ij} x_{ij} + p_{kj} x_{kj} \qquad k \in K_{i_n} \tag{4.27}$$

$$\text{for} \quad i = 1, 2, \ldots, n \qquad k = 1, 2, \ldots, n \qquad j = 1, 2, \ldots, m$$

$$z_{ikj} \in \{0, 1\} \qquad \text{for } i = 1, 2, \ldots, n \qquad k = 1, 2, \ldots, n \qquad j = 1, 2, \ldots, m \tag{4.28}$$

K_{i_n} and I_{i_n} are two notations used between constraints (4.24) $-$ (4.27) that need to be satisfied for the priority sequence condition. The condition I_{i_n} in constraints (4.25) satisfy the situation where if $k = i_a$ there must be $I_{i_a} \neq \{i_{a+1}, \ldots, i_n\}$. The details are as follows:

$$\begin{aligned}
&\text{for} \quad k = i_1, I_{i_1} = \{\} \\
&\text{for} \quad k = i_2, I_{i_2} = \{i_1\} \\
&\text{for} \quad k = i_3, I_{i_3} = \{i_1, i_2\} \\
&\text{for} \quad k = i_n, I_{i_n} = \{i_1, i_2, \ldots, i_{n-1}\}
\end{aligned}$$

While for constraints (4.24), (4.26) and (4.27), we have a situation where $i = i_a$, the set K_{i_a} must satisfy: $K_{i_a} \neq \{i_1, \ldots, i_a\}$. The details are as follows:

Task J_i	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9	J_{10}
M_1	13	20	16	10	19	9	11	18	16	20
M_2	19	5	12	11	15	17	18	5	4	13

Table 4.2: Tasks information for the small example problem

$$\text{for} \quad i = i_1, K_{i_1} = \{i_0, \ldots, i_n\} \ /\{i_1\}$$

$$\text{for} \quad i = i_2, K_{i_2} = \{i_0, \ldots, i_n\} \ /\{i_1, i_2\}$$

$$\text{for} \quad i = i_3, K_{i_3} = \{i_0, \ldots, i_n\} \ /\{i_1, i_2, i_3\}$$

$$\text{for} \quad i = i_n, K_{i_n} = \{i_0, \ldots, i_n\} \ /\{i_1, \ldots, i_n\} = i_0$$

4.2.4 A Small Example

In the following we present a small example to illustrate what has happened in the constraints that satisfies the priority conditions. It aims to give in brief the selection that needs to be satisfied for the priority sequence condition and assignment condition in the constraints. The processing time of 10 tasks for two unrelated parallel processors are given in Table 4.2. The sequence of priority list is $J_5 - J_8 - J_2 - J_7 - J_6 - J_1 - J_{10} - J_3 - J_9 - J_4$. The problem of $R|priority(general)|C_{max}$ is formulated as the above MILP model and then solved by the AIMMS 3.10 software package which uses CPLEX 12.1 as the solver for minimizing the makespan. The values for I_{i_n} and K_{i_n} that are included in constraints (4.25), (4.24), (4.26) and (4.27) are shown in Table 4.3. The optimal solution of the task sequence and the completion time for every processor are given in Table 4.4.

Therefore, from the table the objective value is $C_{max} = max\{C_1, C_2\} = C_1 = 52$. The solution to the model was found in a time of 0.01 seconds.

Priority List i_n	Task J_{i_n}	I_{i_n}	K_{i_n}
i_1	J_5		J_0,J_8,J_2,J_7,J_6,J_1,J_{10},J_3,J_9,J_4
i_2	J_8	J_5	J_0,J_2,J_7,J_6,J_1,J_{10},J_3,J_9,J_4
i_3	J_2	J_5,J_8	J_0,J_7,J_6,J_1,J_{10},J_3,J_9,J_4
i_4	J_7	J_5,J_8,J_2	J_0,J_6,J_1,J_{10},J_3,J_9,J_4
i_5	J_6	J_5,J_8,J_2,J_7	J_0,J_1,J_{10},J_3,J_9,J_4
i_6	J_1	J_5,J_8,J_2,J_7,J_6	J_0,J_{10},J_3,J_9,J_4
i_7	J_{10}	J_5,J_8,J_2,J_7,J_6,J_1	J_0,J_3,J_9,J_4
i_8	J_3	J_5,J_8,J_2,J_7,J_6,J_1,J_{10}	J_0,J_9,J_4
i_9	J_9	J_5,J_8,J_2,J_7,J_6,J_1,J_{10},J_3	J_0,J_4
i_{10}	J_4	J_5,J_8,J_2,J_7,J_6,J_1,J_{10},J_3,J_9	J_0

Table 4.3: Value I_{i_n} and K_{i_n} considered in constraint

Processor M_j	Optimal Task Sequence	Completion time C_j
M_1	$J_5 - J_7 - J_6 - J_1$	52
M_2	$J_8 - J_2 - J_{10} - J_3 - J_9 - J_4$	50

Table 4.4: Optimal solution for the example problem

4.3 Computational Experiments

This section carries out a comprehensive computational testing of our MILP model to see how well the model performs. First, we implement a case study to validate the performance of the model. Next, we design and generate simulations for computational testing to evaluate the model. Then, we report and analyze the result of the MILP.

4.3.1 A Case Study

This case study is intended for experience the whole process of a medium sized problem. The case study is taken from Wu and Ji (2009). The case study consists of 46 tasks for a printed circuit board (PCB) assembly production operation. The task needs to be assigned to five assembly lines called Line 1, Line 2, Line 3, Line 4 and Line 5. These five assembly lines represent the five unrelated parallel machines. The job data for the PCB is shown in Table 4.5. The processing time is displayed for each line and the time values in the table are shown in hours. The time is set to $1,000$ if a job cannot be processed on the line. The priority list of the job on the assembly lines are defined on ready times and the due date to suit our problem.

The MILP model has been implemented and compiled using the CPLEX 11.0 package. The MILP gives the optimum result for the instance problem. The result of the makespan is 123.01 hours and the problem is solved within 83.44 seconds. The completion time for each machine is $C_1 = 122.74$, $C_2 = 122.86$, $C_3 = 122.88$, $C_4 = 123.01$ and $C_5 = 122.98$. Therefore the result of the makespan is 123.01 hours on M_4. Figure 4.2 shows the Gantt Chart of task scheduling obtained by MILP.

Priority list L	Task no. J_i	Line 1	Line 2	Line 3	Line 4	Line 5
L_1	J_{31}	1000	16.42	21.83	17.81	17.81
L_2	J_{39}	35.17	33.87	36.74	30.92	30.92
L_3	J_2	13.62	13.2	13.2	12.75	12.75
L_4	J_1	1000	11	11	1000	1000
L_5	J_7	24.51	22.75	24.68	22.75	22.75
L_6	J_{27}	8.96	8.08	8.29	8.08	8.08
L_7	J_{33}	9.56	1000	11	1000	1000
L_8	J_{28}	10.85	9.41	10.59	9.41	9.41
L_9	J_{34}	15.98	15.48	15.21	1000	1000
L_{10}	J_{36}	11.65	11.45	11.66	11.45	11.45
L_{11}	J_5	17.48	15.12	15.12	15.12	15.12
L_{12}	J_6	23.64	18.89	18.89	26.31	26.31
L_{13}	J_{32}	1000	16.42	21.83	17.81	17.81
L_{14}	J_{11}	8.89	6.5	7.52	1000	1000
L_{15}	J_{40}	16.21	19.98	15.75	15.75	15.75
L_{16}	J_9	15.23	12.75	15.14	12.75	14.75
L_{17}	J_{29}	7.68	7.59	7.77	7.12	7.12
L_{18}	J_{12}	14.23	11.89	1000	1000	1000
L_{19}	J_{25}	11.82	10.55	10.36	10.55	10.55
L_{20}	J_{22}	5.36	4.98	4.68	1000	1000
L_{21}	J_{14}	12.5	12.22	11.73	11.78	11.78
L_{22}	J_3	15.89	14.49	14.49	13.87	13.87
L_{23}	J_{30}	7.93	7.47	7.2	6.89	6.89
L_{24}	J_{35}	14.87	12.89	16.94	1000	1000
L_{25}	J_8	24.51	22.75	24.68	22.75	22.75
L_{26}	J_{20}	11.74	11.45	11.66	11.45	11.45
L_{27}	J_{26}	13.03	11.77	10.74	11.77	11.77
L_{28}	J_{10}	15.23	12.75	15.14	12.75	14.75
L_{29}	J_{13}	14.23	11.89	1000	1000	1000
L_{30}	J_{18}	8.66	8.32	1000	8.32	8.32
L_{31}	J_{24}	10.56	1000	10	1000	1000
L_{32}	J_{19}	8.66	8.32	1000	8.32	8.32
L_{33}	J_{15}	12.74	9.36	9.66	11.56	11.56
L_{34}	J_{43}	18.46	18.34	19.29	16.69	16.69

Priority list L	Job no. J_i	Line 1	Line 2	Line 3	Line 4	Line 5
L_{35}	J_{45}	27.45	1000	1000	22.92	22.92
L_{36}	J_{41}	45.78	39.55	40.33	37.81	37.81
L_{37}	J_{21}	11.32	11.22	11.35	11.24	11.24
L_{38}	J_{23}	9.84	7.62	11.62	1000	1000
L_{39}	J_{16}	15.84	13.24	1000	13.24	13.24
L_{40}	J_{37}	11.39	11.22	11.35	11.24	11.24
L_{41}	J_{46}	27.45	1000	1000	22.92	22.92
L_{42}	J_{42}	6.74	5.87	6.23	4.85	4.85
L_{43}	J_{44}	6.88	7.02	1000	6.59	6.59
L_{44}	J_4	15.89	14.49	14.49	14.64	14.64
L_{45}	J_{17}	15.84	13.24	1000	13.24	13.24
L_{46}	J_{38}	9.37	1000	1000	8.82	8.82

Table 4.5: Tasks information for the case study

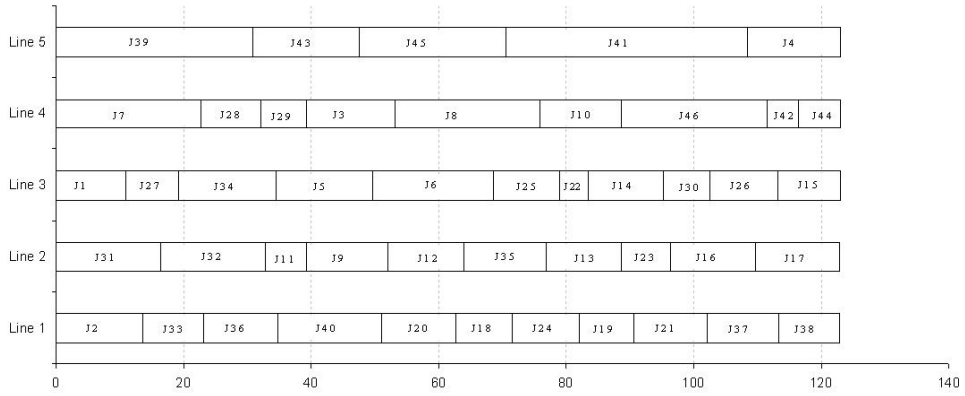

Figure 4.2: Gantt chart for the case study

4.3.2 Experimental Design

We implement our MILP model using AIMMS 3.10 on a personal computer (PC) with Intel Core 2 2.66 GHz 1.95 GB RAM. The simulation data for the problem $R|priority(general)|C_{max}$ is generated as follows:

1. The number of independent tasks are n=$\{20, 30, 40, 50, 60, 80, 100\}$.
2. For every set of tasks, we use a different number of processors $m = \{2, 3, 4, 5\}$. In total, there are 28 combinations of m and n.
3. For every combination of m by n, we generate 20 instances. Therefore, the total number of instances that we have are $28 \times 20 = 560$. We use the instance generator that we create in AIMMS by computing the procedure as follows:
$$p(i,j) := round(Uniform[p^{min}(i,j), p^{max}(i,j)])$$
We use an interval for the processing time where $p^{min}(i,j) = 1$ and $p^{max}(i,j) = 60$.
4. The order of the priority is generated to be the same for all sets of instances to make it comparable.

4.3.3 Computational Results

We now present our result of the MILP model. For this MILP model, the CPU times are reported for every instance. We also consider the result with and without computation time limit. Note that all solutions have been observed and all instances have been solved up to optimality, i.e 0% gap, even for the cases with computation time limit. The result for the model will be presented as a gap (%) and can be calculate as follows:

$$Gap(\%) = \frac{C_{max} - C^*_{max}}{C^*_{max}} \times 100 \tag{4.29}$$

where C^*_{max} is the optimum solution and C_{max} is the value obtained when reached the specified time limit

Computation time in dependence of the problem size Table 4.6 shows for each combination of $n \times m$ the average computational times in seconds. In the table, we also present the maximum and the minimum CPU time for the n tasks on the m processors. In the simulations, we are interested in the evaluation of performance in terms of CPU time for the model in computation of exact optimal solution for the $R|priority|C_{max}$. As we can see from the table, this MILP model obviously can be solved within a reasonable time for all tested cases and guaranteed the obtained solution is optimal. The average reported in the table for every combinations are less than 1 hour.

In average column, we also report in bracket the total instances solved with total computational time limit of 1 hour. From the 560 instances of all combination, only six instances are over the time limit i.e 1.07%. Specifically, there are two cases obtained from the 100 tasks by 4 processors and another four cases are from 100 by 5 processors. For more detail, in Table 4.8 we present for each involved test cases the minimum gap and the maximum gap at the time limit. For example, from a further observations on the cases of 100 tasks by 4 processors, the both cases that reached the time limit have obtained the best solution with the gap less than 1% since no later than 150 seconds of the CPU time. Overall, the instances for the six cases that reached the stopping criteria have the gaps less than 0.9% at the time limit. We can conclude that for these cases, the model perform well for obtained the result with small gap on less computation time.

Computation time of instances with fixed number of tasks Figure 4.3 presents the average computation time of processors while tasks stay constant. The diagrams

Number of Processor m	Number of Task n	Average	Minimum	Maximum
2	20	0.071	0.03	0.14
	30	0.134	0.08	0.2
	40	0.282	0.16	0.42
	50	0.5385	0.28	0.8
	60	0.8695	0.47	1.55
	80	1.8915	0.86	3.52
	100	3.754	1.36	10.13
3	20	0.299	0.08	0.52
	30	2.041	0.17	11.38
	40	3.6245	0.56	27.3
	50	5.2585	1.28	21.22
	60	6.5375	1.31	30.2
	80	14.826	3.59	33.94
	100	24.256	4.22	91.58
4	20	0.6025	0.22	1.47
	30	18.498	1.38	153.41
	40	24.15	3.23	107.75
	50	29.7035	2.14	376.98
	60	80.519	3.98	1089.33
	80	201.951	4.53	1408.11
	100	794.1135 (2)[1]	25.64	5233.48
5	20	2.146	0.27	12.2
	30	28.6445	1.27	381.41
	40	55.1645	2.75	233.78
	50	62.7035	3.88	305.41
	60	95.0345	9.92	245.16
	80	289.7335	42.41	1583.86
	100	2377.752 (4)[2]	47.86	12720.42

Table 4.6: CPU times for the MILP model (in seconds).[1,2] Instances solved over the time limit of 1 hour

Number of Processor m	Number of Task n	Instances over the time limit	Minimum gap	Maximum gap
4	100	2	0.27%	0.34%
5	100	4	0.41%	0.90%

Table 4.7: Instances solved over the time limit of 1 hour

indicate on x-axis by the number of processors and the y-axis are the CPU time. The objective is to evaluate the model when the system is growing and the model has more choices in assigning the tasks. The experiment show that the CPU times have been longer when there are more resources to allocate. The model increases the search iterations until optimal value. Furthermore, the model has to allocate and schedule the task according to the priority consideration that need to be satisfy and may take more iterations. All the graphs have a moderate increase in the plot size. Here, for the cases $n = 20$, $n = 30$, $n = 40$ and $n = 50$ the graph exponentially increased as the system having more processors due to the time-consuming memory operations. However, there are interesting observations in the figure for the cases of $n = 60$ and $n = 80$ when the plot is having only slight increase after $m = 4$. The observations show us that as the number of tasks and processors increase, the performances of model improve when the model has more resources to allocate and increase the possibilities for tasks assignment.

Computation time on different stopping criteria of computation time Figures 4.4 $-$ 4.7 show for each processor, the percentage of the solved instances for every time limit. The experiment is performed to have a closer observation from the results obtain in table 4.8. Therefore, the experiment is to evaluate the quality of the performance based on completion time for the MILP model in obtaining the optimal result. The MILP model are tested on 10 different range of time limits (in seconds) as follows : 5, 10, 30, 60, 150, 300, 600, 900, 1800 and 3600. Table 4.8 is the average percentage for the solved instances that showed in figures 4.4 $-$ 4.7.

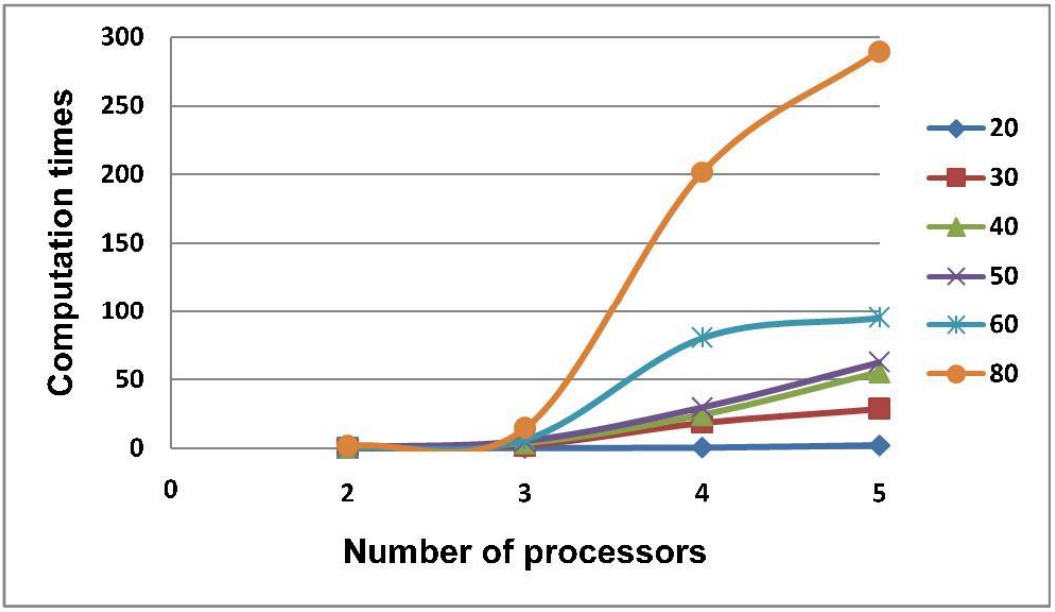

Figure 4.3: Computation time over the processors with constant number of tasks

Time Limit	5	10	30	60	150	300	600	900	1800	3600
Average (%)	52.1	61.3	75.2	82.3	90.9	94.8	95.9	96.6	98.6	98.9

Table 4.8: The average percentage of solved instances for $m = \{2, 3, 4, 5\}$ at different time limits

Here, we can conclude that the MILP model performs well. Note that the 52.1% of the total instances have been solved within 5 seconds. Moreover, 90.9% of the instances have reach optimal solution by the 150 seconds. Only 9.1% instances have CPU time after 150 seconds. In other words, the model is very efficient and can be solved with less amount of computation times.

4.4 Summary

In this chapter, we introduced a special case of priority consideration for the task characteristic. The new feature is applied to the unrelated parallel processor system in minimizing the makespan. We refer the problem as $R|priority|C_{max}$. We

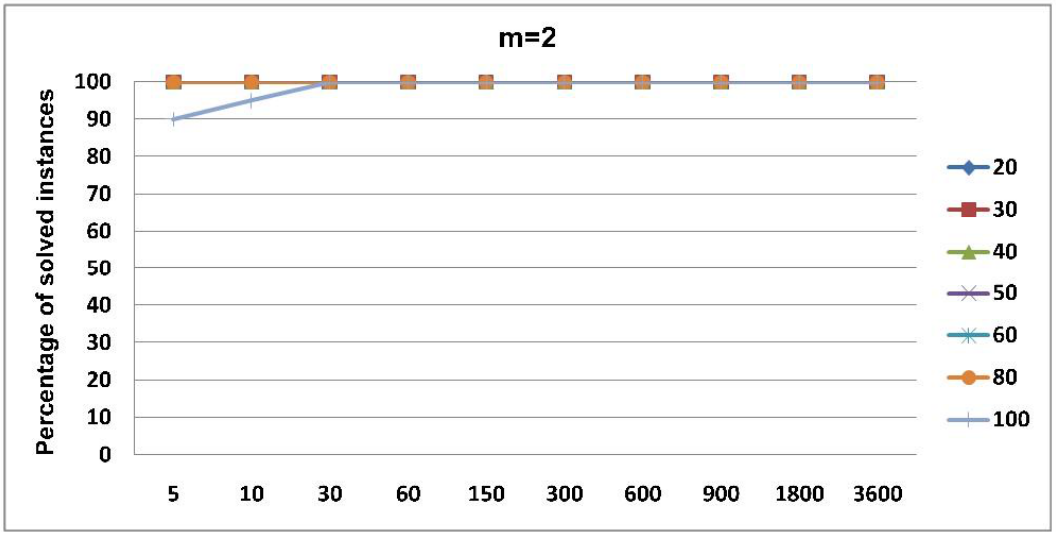

Figure 4.4: The percentage of solved instances for $m = 2$ at different time limits.

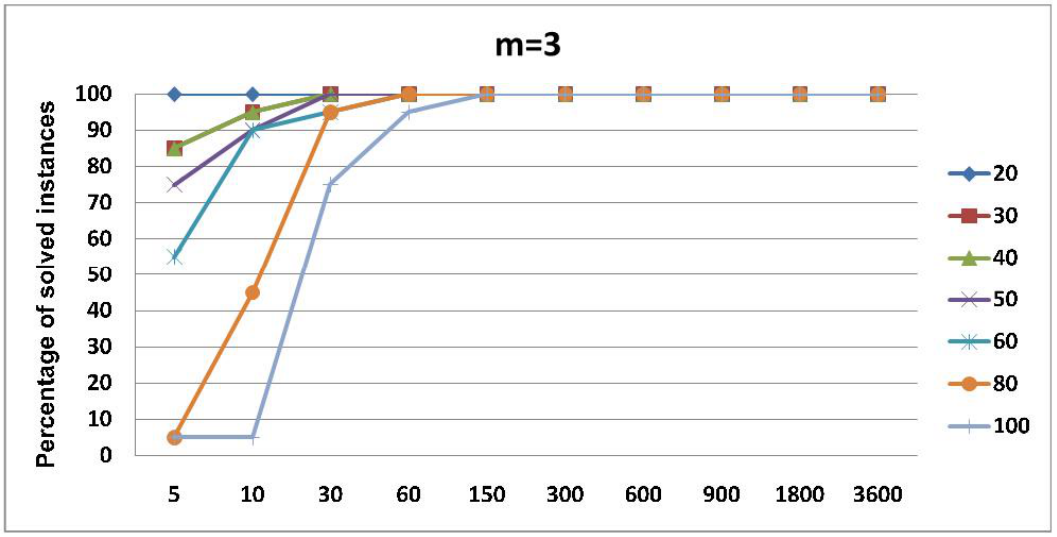

Figure 4.5: The percentage of solved instances for $m = 3$ at different time limits

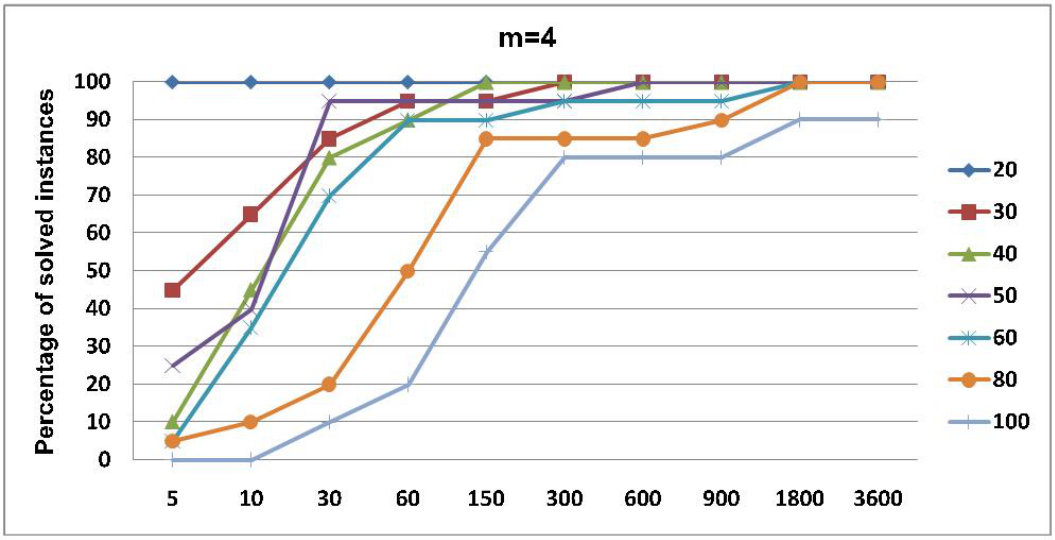

Figure 4.6: The percentage of solved instances for $m = 4$ at different time limits

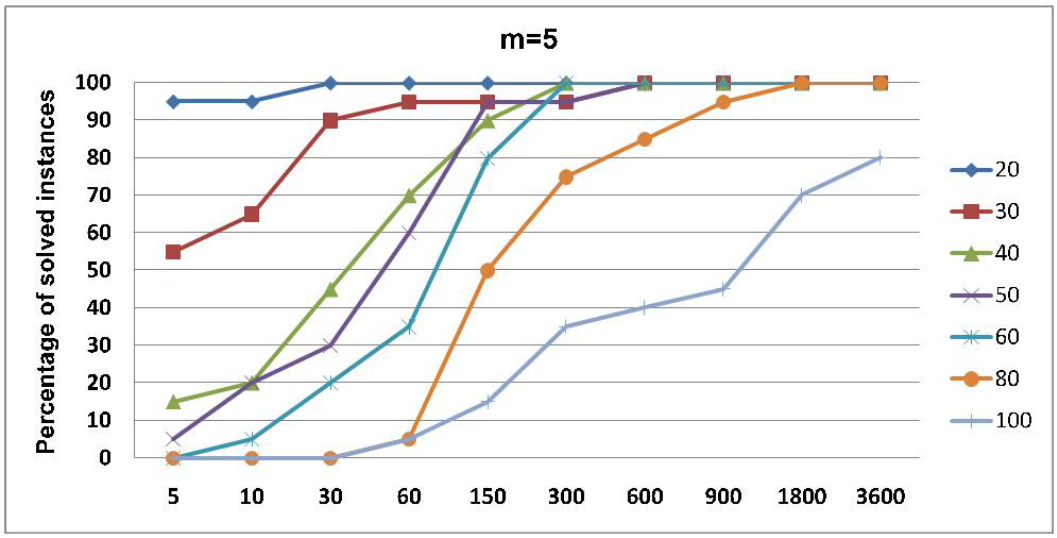

Figure 4.7: The percentage of solved instances for $m = 5$ at different time limits

develop mixed integer liner programming (MILP) model to solve the problem. We consider three different order of task sequence in priority consideration: ascending order, descending order and general priority list. The MILP model have been implemented and produced 100% optimum solutions from the total 560 instances with 28 different combinations of size problem. Examination of the performance in CPU time of the model show amazing results with the maximum average solution for every problem size is obtain by 100 tasks by 5 processors within 39.6 minutes. The MILP model is shown to be effective for solving $R|priority|C_{max}$ and obtain optimum result within a short time.

Chapter 5

Resource Disruption in Parallel Processor System

The previous chapter concentrates on the scheduling problem without having any interruptions in the system. In this chapter we address a feature where interruptions that have occurred. The situation happens when the availability of the parallel processors in the time slot decreases in certain time periods and we define the problem as resource disruption. It is necessary to consider a recovery option for this issue in the scheduling plan to overcome the possibilities of having infeasibility of the original plan. Our approach for the recovery is task rescheduling which is to reassign the tasks in the initial schedule plan to reflect the new restrictions. A recovery mixed integer programming (MIP) model is proposed to solve the disruption problem for two cases of disruptions: predictive disruption and post-disruption problem.

The contents of this chapter is as follows. In Section 5.1, we discuss the problem description for the disruption problem. In Section 5.2, we give the recovery MILP model for several cases in disruption problem. We implement and test the model in Section 5.3. Then, conclude with a brief summary of the chapter in Section 5.4.

5.1 Problem Descriptions

The issue presented in this chapter is a disruption situation that arises in the task scheduling problem on parallel processor systems. Initially, we consider a disruption when there are changes in the operating system that might need to have a back up plan in order to complete the task as schedule in the original plan. Disruption results in the need to the scheduling problem.

There are many internal and external factors that cause the disruptions. For example, the internal factors that contribute to the disruptions are power supply failure, need for machine maintenance, issue in the quality control requirement, raw materials shortage, delay in producing the target and sickness in personnel. There are also external disruptions factors that can occur and affect the performances of the plan. The changes in the weather, changing customer orders, the cancellation/delay from the suppliers, political issues and the new government polices are several reasons that may cause the external disruptions.

The disruption in the system will change the system environment for the scheduling. A disruption problem might impact on the original schedule and in the worse case cause it to become infeasible. Therefore, it is necessary to have a recovery decision. The recovery decision should determine the possible options so that the problem could be solve optimally. A new recovery objective and constraints are important to represent the recovery problem.

In order to address this case, an optimal operation decision has to be made in the face of possible disruption. There are options that are normally considered for the recovery: rescheduling, mode alternative (eg. subcontracting and activity cancelation) and resource alternative (Zhu et al.,2005). Our recovery decision is rescheduling because we aim to response to the original schedule and keep changes at a minimum level without any additional cost of alternative option if possible.

The following definition, classifies the disruption problem in area of disruption management:

Definition 5.1 (*Disruption management* (Yu and Qi, 2004)) At the beginning of a business cycle, an optimal or near-optimal operational plan is obtained by using certain optimization models and solution schemes. When such an operational plan is executed, disruptions may occur from time to time caused by internal and external uncertain factors. As a result, the original operational plan may not remain optimal, or even feasible. Consequently, we need to dynamically revise the original plan and obtain a new one that reflect the constraints and objectives of the evolved environment while minimize the negative impact of the disruption. This process is referred to as disruption management.

Figure 5.1 provides a summary of the scenario for the disruption problem in the scheduling and leads us to our approach of rescheduling of the initial plan. Note that rescheduling and scheduling are different in some aspects. In scheduling, we have enough time to plan the strategies in advance and in detail. While in rescheduling, there is a time restriction in the decision making for the real practice to over come the disruption. The goal for the scheduling problem can be a plan to have nearly optimum or an optimum results and for the rescheduling problem, it is often only to aim for a feasible schedule that can reduce the major disruption. Normally, the scheduling problem might have only one approach required compare to the rescheduling problem may have multiple options with different considerations.

We focus the rescheduling on unrelated parallel processor system. In the system, we are concerned with the interruption during the scheduling, a machine becomes unavailable at a certain time. The problem is specifically described as resource disruption. The resource disruption also refers to the machine shortage in the parallel processor system that gives impact to the task scheduling. The potential ma-

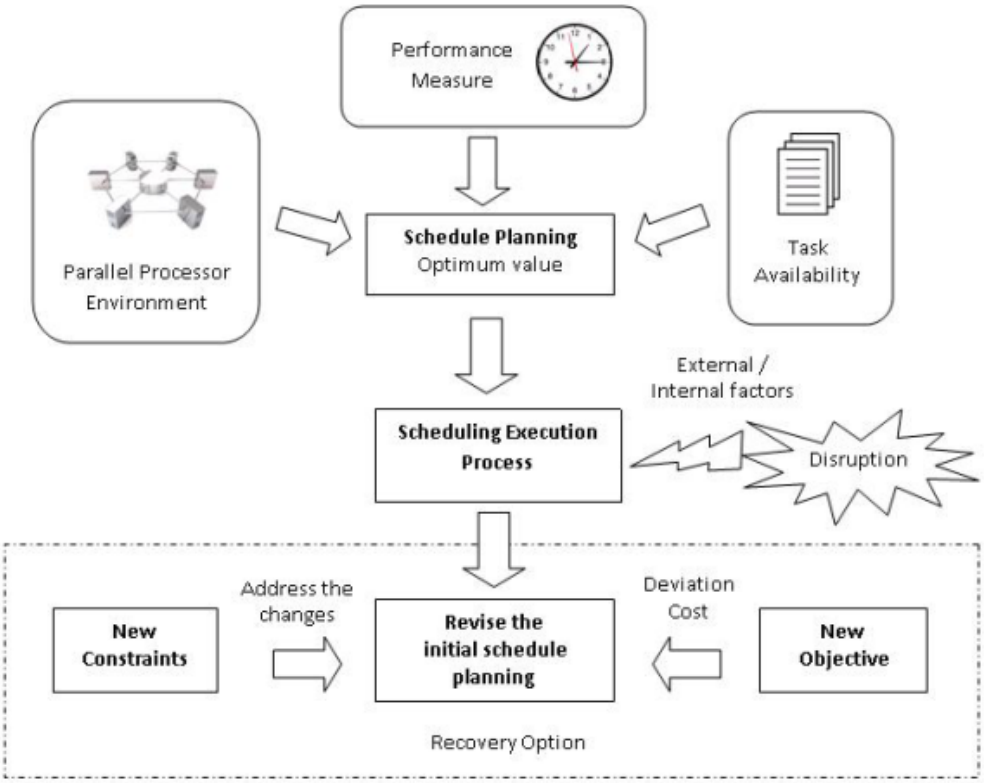

Figure 5.1: Disruption management process

chine disruption are considered in which the unavailability can be predictive. The time of disruption and the duration can be expected due to managerial arrangement such as scheduled machine maintenance or recession of employee. This situation is called predictive disruption. The case of prediction disruption is more informed. Therefore, the schedule can be revised before the disruption situation really happens. If the case of disruption is not known in advance and occurred unpredictably, it is considered as post-disruption. The rescheduling for recovery option can only start on or after the disruption period. This post-disruption policy in-charge of the remaining tasks left in the system for the revision process.

Disruption management is very crucial and has been applied to many real practice environments such as in the semiconductor industry, logistics industry and the most active in this disruption applied area is the airline industry. For instance, Clausen at al. (2010) investigated the airline recovery problem for scheduling involving the aircraft routing, the cabin crew scheduling and the passenger recovery. Dorndoft et al. (2007) presented a recovery strategies for flight gate assignment which include the activities of the landing aircraft, departure gate and the parking lane. The flight delay is one of the causes that need a flight gate recovery to recontract the airline schedule. The improvement of the operation due to the disruption has a considerable impact and benefit. The recovery operation could reduce the airline cancelation, the delay of other aircraft and to save cost on the other disruption that might occur. Kohl et al. (2007) gave an overview of the airline disruption management in various aspect including the network structure, resource planning, recovery strategies and also the operation control. They also reported about the experience from a development project that have been done. The project evaluated the recovery system that involved dedicated passenger recovery system, dedicated aircraft recovery solver, dedicated crew recovery solver and integration recovery.

In the next section, we focus on our rescheduling MILP model for the parallel processor system during the occurrence of resource disruption problem.

5.2 Mixed Integer Programming Model for Resource Disruption

We consider the system that has n independent tasks, J_i ($i = 1, 2, \ldots, n$), and m unrelated parallel processors, M_j ($j = 1, 2, \ldots, m$), but there are disruption features. Both sets of tasks and processors are all assumed available at time zero. Even there are disruption in the system, we still assume that the tasks are non-preemtive where the tasks need to be processed without interruption. Therefore, if the disruption occurs in the middle of a processing task, the task should be rescheduled from the start which this is called a non-resumable case. The task can be assigned to any processor and the task migration between processor is allowed for the reschedule process. Each processor is unavailable when the system is having disruption. Once the disruption has been fixed, the processor is immediately available until the system satisfies the stopping criteria. Each processor is capable to support one task at a time by an integer processing time except during the disruption activity.

We study the existence of disruption on unrelated parallel processor scheduling problem. The disruption occurs due to a resource shortage where one of the parallel processors is facing breakdown problem during the task allocation which give impact to the initial plan. Our objective is to reschedule the original unrelated parallel processor scheduling due to resource disruption that minimizes the makespan. Using the three field notation of Graham et al. Graham *et al.* (1979), we refer this problem as $R|disruption|C_{max}$. We develop a MILP assignment model to solve the problem. This section describes the developed model for the recovery scheduling of disruption for two type of disruption policies: predictive disruption and post-disruption.

The Figure 5.2 presents our approach in design our mathematical formulation

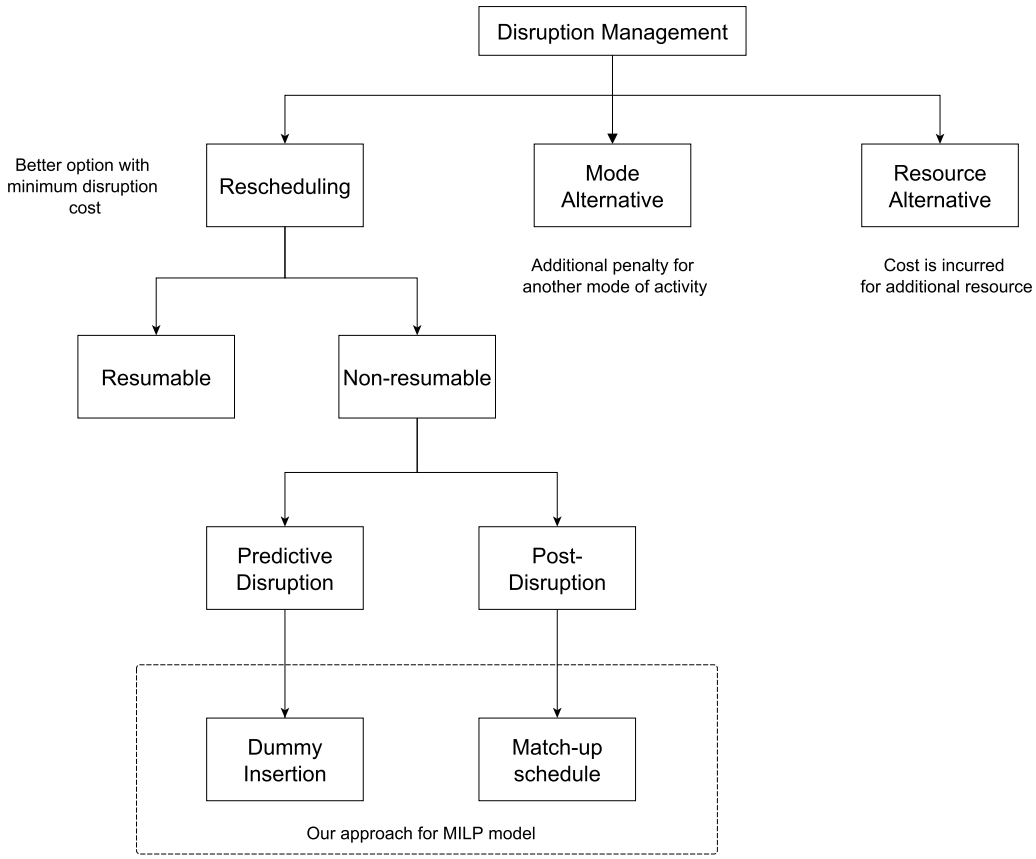

Figure 5.2: Our MILP model for predictive disruption and post-disruption policy

in solving the predictive disruption and post-disruption problem that consider in this section.

5.2.1 Predictive Disruption Management

Predictive disruption management allows the scheduler to revise the original plan whenever needed to cope with the disruption in the future. The action is immediately taken once the critical scenario already identified happens during the scheduling process. Within a limit of time between the notice of disturbance and the execution, the initial plan needs to generate a recovery operation to reflect with the changes that will appear in the system. We consider a case where the disruption slot is not fixed to prescribed time slot. Therefore, we manage to arrange the optimize time for the disruption using our MILP model with dummy insertion.

Recovery model with dummy insertion

The assignment for the recovery model could allocate the task with a sequence as in the original schedule but with the insertion of disruption slot declared as *dummy task*, i_d ($d = n+1, \ldots, N$), where i_d is the additional set contain in the original task set i. The set of dummy tasks run on the disrupted processor j_d with duration of $p_{i_d j_d}$. The existence of i_d is to utilize the slack occurred in the processor and hence, manage to reduce the maximum of the total completion time. In addition, the model force the disruption not to interfere at the task that is in progress.

The following mathematical model presents the described problem with our decision variables defined as:

$$x_{i'j'} = \begin{cases} 1, & \text{if task } i' \text{ is assigned to machine } j' \\ 0, & \text{otherwise.} \end{cases} \tag{5.1}$$

where $i' = 1, 2, \ldots, n, n+1, \ldots, N$, $j' = 1, 2, \ldots, m$ and $x_{i'j'}$ is an $0-1$ assignment variables.

The objective function requires the minimization of the maximum of the total completion time for processors, that is,

$$\min \qquad y = \max\{C_{j'}\} \tag{5.2}$$

The objective function above has a set of constraints as stated below:

$$\sum_{j' \in \{j, j_d\}} x_{i'j'} = 1 \qquad \text{for } i' = \{1, 2, \ldots, n, n+1, \ldots, N\} \in (i \cup i_d) \tag{5.3}$$

$$\sum_{j' \in \{j, j_d\}} p_{i'j'} x_{i'j'} \leq y \qquad \text{for } i' = \{1, 2, \ldots, n, n+1, \ldots, N\} \in (i \cup i_d) \tag{5.4}$$

$$y_j = \sum_{i' \in \{i, i_d\}} p_{i'j'} x_{i'j'} \qquad \text{for } j' = \{1, 2, \ldots, m\} \in (j \cup j_d) \tag{5.5}$$

$$y_j \leq y \qquad \text{for } j' = \{1, 2, \ldots, m\} \in (j \cup j_d) \tag{5.6}$$

$$y \geq 0 \tag{5.7}$$

$$x_{i'j'} \in \{0, 1\} \quad \text{for } i' = \{1, 2, \ldots, n, n+1, \ldots, N\} \in (i \cup i_d) \tag{5.8}$$

$$\text{and for } j' = \{1, 2, \ldots, m\} \in (j \cup j_d)$$

Constraints (5.3) ensures that each task in set $(i \cup i_d)$ is assigned to only one of the m machines. Constraints (5.4) ensure that the total completion time of task i' is restricted with makespan. Constraints (5.5) and (5.6) are additional constraints to the model to define the total completion time of processor j' and bounded by makespan. Constraints (5.7) and (5.8) requires the non-negative makespan and binary assignment.

An example of information for the unrelated parallel processor problem with one disruption insertion which denoted as J_{d_1} are shown in Table 5.1. This problem consist of 10 tasks with 1 insertion of dummy task that represent the disruption process and all the tasks will be assign on 3 unrelated parallel processor system. From the table, only one resource disruption will occur for 4 time units on machine 2 and not applicable to other machines in the system. An optimal initial schedule for J_1 until J_{10} is presented in Figure 5.3. The number in the bracket is the processing time and the number next to it is the task number. The start time and the completion time for every task also stated in the figure. The value for the makespan of the initial

Task Number J_i	Processing Time Processor 1	Processor 2	Processor 3
J_1	9	8	5
J_2	4	3	3
J_3	7	2	8
J_4	6	5	5
J_5	2	7	4
J_6	10	8	3
J_7	8	3	7
J_8	8	9	10
J_9	7	10	1
J_{10}	9	8	3
.....	.....	.....	.....
J_{d_1}	na	4	na

Table 5.1: An example for dummy insertion approach

schedule is 13 time units.

When a resource disruption is scheduled to occur in the machine 2 (denoted as J_{d_1} in this example), our MILP model has to produce a rescheduling process to adopt the disruption. The approach is to insert the disruption slot as a dummy task on the M_2 and obtained an optimal condition for the disruption insertion. In the Figure 5.4, the dummy task has four options to insert the disruption slot on M_2. The option for the sequence are $(J_{d_1}-J_3-J_4-J_7)$, $(J_3-J_{d_1}-J_4-J_7)$, $(J_3-J_4-J_{d_1}-J_7)$

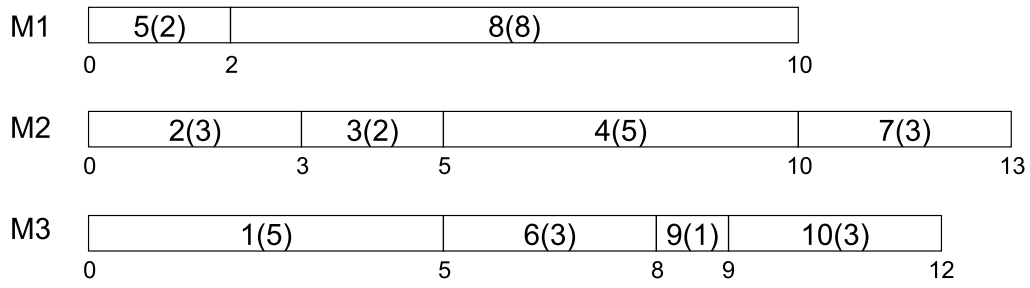

Figure 5.3: Initial schedule with $C_{max} = 13$

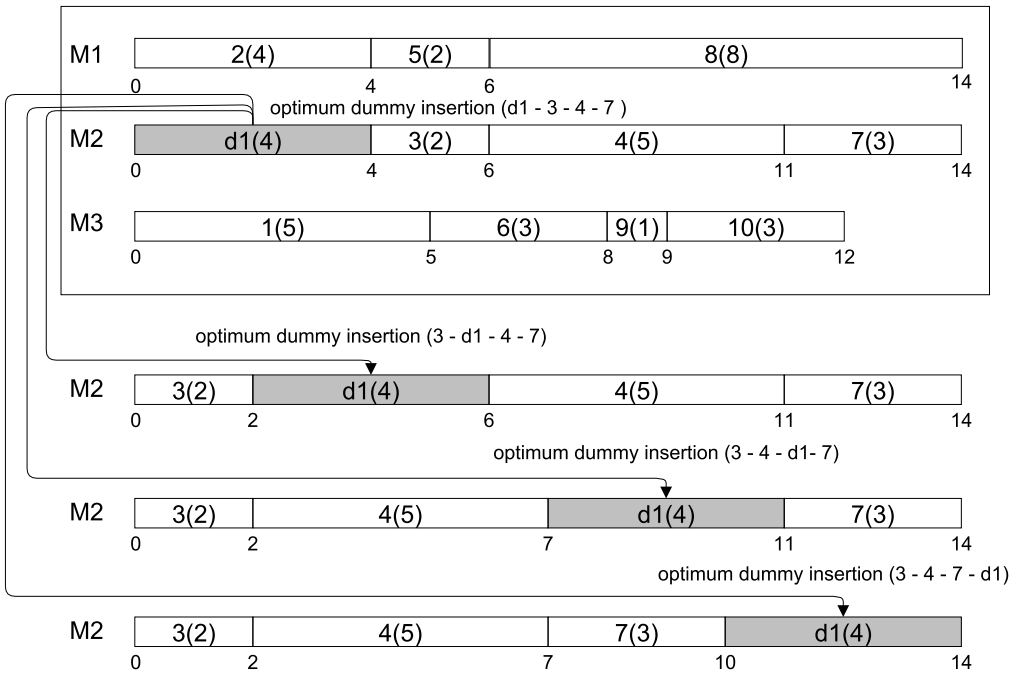

Figure 5.4: Rescheduling process with dummy insertion options obtained $C_{max} = 14$

and $(J_3 - J_4 - J_7 - J_{d_1})$ as stated in the figure. The new makespan obtained by the rescheduling model is 14. The different between the makespan obtained by initial schedule and the repair schedule is $14 - 13 = 1$ time unit only.

5.2.2 Post-disruption Management

In this section, we consider a post-disruption management for resource disruption. Post-disruption problem is a case where the disruptions occur during the execution of the initial schedule. We can only know the disruption event until the disruption really enters the system. All the remaining tasks in the system which are not completed will be send for rescheduling. The tasks that are already processed are not involved in the reschedule. In the rescheduling model, we define a recovery objective and recovery constraints for the match up strategy to revise the remaining tasks in the system and stay close to the initial schedule during the resource disruption problem.

Recovery Objective

The recovery objective is for the rescheduling model to define a new performance measure and check the stability of the new schedule.

Deviation costs

In this case, the cost associated with the deviation between the original and the new schedule is taken into account. The idea of introducing deviation costs is to construct a new schedule to stay close to the original operational plan. Therefore, the disruption cost occurred due to the changes of the plan could be reduced. The deviation cost involved in our operation is the migration task to another processor that is different from the initial schedule. If there are no changes in the rescheduling for this criteria, no deviation cost will be applied.

In order to consider the migration tasks, we let the D_{ij} be is the migration tasks i from processor j to another processor after the disruption occurred and can be written as follows:

$$D_{ij} = \phi_{ij} - x_{ij} \tag{5.9}$$

where the ϕ_{ij} is the assignment for task i on processor j for the initial schedule and x_{ij} is the variables for the current schedule for task i that processing on processor j in the rescheduling phase. The total deviation cost in terms of the total migration task is $\sum_i \sum_j D_{ij}$.

Multi-criteria decision making

A disruption problem also can be modeled as multi-criteria decision making to consider not only the new environment, but also the original goal of the operational

plan. The purpose of considering the original and the current objective function is that the original schedule has made many preparations (such as in terms of material, equipment, customers and the milestones). The changes in these preparations may bring additional disruption costs. Therefore, we can address the disruption problem with the following objective function:

$$\min \qquad Q = \omega_1 y + \omega_2 z \tag{5.10}$$

where ω_1 and ω_2 are given weights with $\omega_1 + \omega_2 = 1$. In our model, y is the original performance measure which is makespan where $y = max\{C_j | j = 1, \ldots, m\}$ and z is the total deviation costs, $z = \sum_i \sum_j D_{ij}$.

Recovery Constraint

We are going to adopted match-up strategies (Moratori et al., 2008) for our rescheduling approach. Our match-up rescheduling algorithm consists of the following main steps:

Step 1: The initial schedule of the original plan that is currently implemented in the system is identified.

Step 2: The recovery start time is determined by t_s where t_s is the start time for the rescheduling.

Step 3: A new MILP rescheduling model is developed (i.e defined in (5.11) until (5.19)), starting from t_s to obtain an optimum schedule.

Step 4: The performance and the stability of the model is verified.

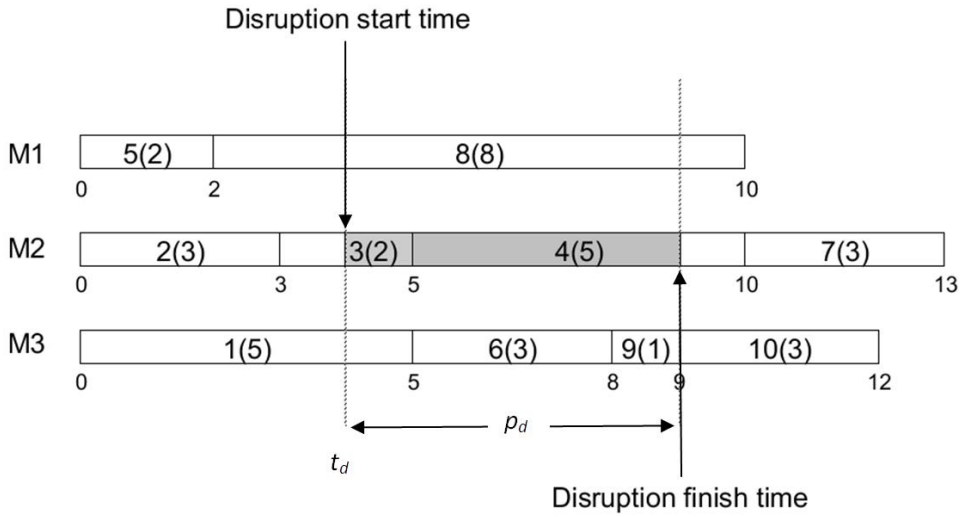

Figure 5.5: The disruption period

The match up strategy for the rescheduling is to handle the disruption task in the system when the resource disruption has occurred. In order to accommodate the remaining tasks, we have to set the recovery start time. The recovery constraints are developed for the recovery start time onwards. Therefore, to implement the rescheduling MILP model, we first have to determine the rescheduling start time and we assume that the initial schedule is already known.

As an example, we use the same initial schedule as in Figure 5.3 where the initial value of n is 10. Suppose that, during the execution of the initial schedule, there is a disruption on one of the machines at t_d on a certain amount of time, p_d as shown in Figure 5.5. Therefore, all the uncompleted tasks after t_d on the disrupted machine and on other machines have to be rescheduled starting from the t_s. After the start time is determined, the current time index is reset to 0 and the uncompleted tasks in the system are re-index from 1 to n. To identify the t_s for the rescheduling, there are two different approaches that we could use.

The first approach is used in Ozlen and Azizoglu (2011) and can be illustrated as in Figure 5.6. Suppose the initial schedule is being processed until t_d

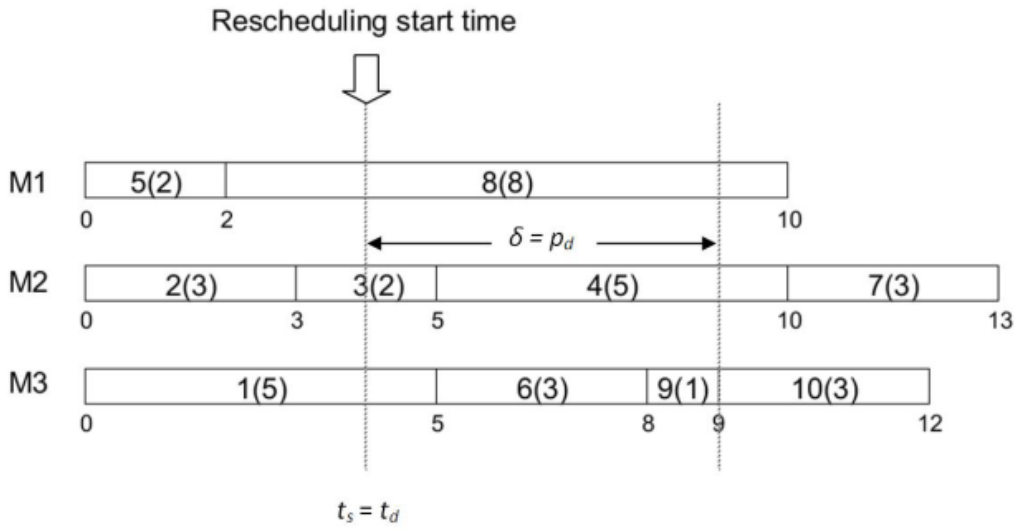

Figure 5.6: Rescheduling start time when $t_s = t_d$

before the disruption occurred. In this approach, all the tasks on the disrupted processor and un-disrupted processor have to be rescheduled start from t_d. Therefore, the reschedule start time, t_s is equal to t_d. Since our problem is assumed to be non-resumable, the uncomplete tasks on any processor that being process on or after t_d have to be restart again. The tasks that involved in the rescheduling process are $\{J_1, J_3, J_4, J_6, J_7, J_8, J_9, J_{10}\}$. The disrupted processor, M_2 is unavailable during the rescheduling process for δ unit time and in this case $\delta = p_d$. The start time of the rescheduling process is reset and become $t_s = 0$. The value of n is now 8.

In Figure 5.7, the rescheduling start time is determined by the task that being processed on the un-disrupted processor with the earliest finish time after t_d (Yu and Qi, 2004). From the example, J_1 and J_8 is currently in progress on M_1 and M_2 respectively at t_d. However, J_1 is finish earlier that J_8. Therefore, the completion time of task 1, C_1, is declared as t_s and we reset it as 0 for the rescheduling process. In this case, the tasks that need to be rescheduled are $n = 7$ and they are $\{J_3, J_4, J_6, J_7, J_8, J_9, J_{10}\}$. The M_2 is still having disruption and unavailable for $\delta = (t_d + p_d) - t_s$ unit time. If there is a case where the disruption is finish earlier than the all progressing processing tasks, the earliest finish time of the progressing

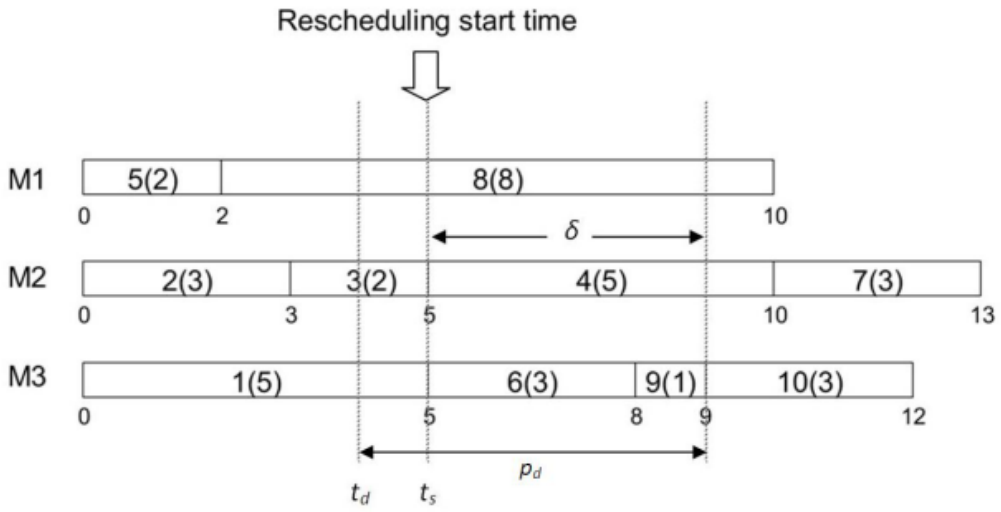

Figure 5.7: Rescheduling start time when $t_s = (t_d + p_d) - \delta$

processing task after $t_d + p_d$ on un-disrupted processor is chosen to be t_s.

After we determine the start time for the rescheduling, we update all the remaining information in the system for the recovery model. For this rescheduling problem, a general model based on time-indexed variables on starting time is introduced. The model define a start time of a task being proposed on a processor. This restriction makes it possible to state the task and the disruption allocation for the problem. The $0-1$ time indexed decision variable is denoted as follows:

$$x_{ijt} = \begin{cases} 1, & \text{if processing of task } i \text{ starts processing on processor } j \text{ at time } t \\ 0, & \text{otherwise.} \end{cases} \tag{5.11}$$

where $i = 1, 2, \ldots, n, j = 1, 2, \ldots, m$ and $t = 1, 2, \ldots, f$.

We call a disrupted processor if it becomes unavailable at certain period of time. The disruption slot has been noticed once the disruption hits the processor. An example is a network down problem in a company. There is a last minute notice about a server down during a certain period from authorized department that they

have undergoing some sudden network operation. As a result, some of the departments may experience limited or no network access to the company network drives. All the operations on the effected area have to be reschedule to face the disruption until the disruption is over.

In the model, we consider the disruption allocation problem and remain the scheduling efficiency measure in minimizing the makespan. To describe the resource disruption that have been allocated, we let A_{jt} is the processor availability when,

$$A_{jt} = \begin{cases} 1, & \text{if the processor } j \text{ is available at time } t \text{ without disruption} \\ 0, & \text{otherwise.} \end{cases} \tag{5.12}$$

The following MILP model contains the recovery constraints set for the rescheduling time horizon where one processor is disrupted at time 0. The assignment variables (5.11) are used in the model and the rescheduling model can be formulated as follows with all information have been updated to reflect the disruption allocation:

$Minimize \qquad Q = \omega_1 y + \omega_2 z$

$s.t$

$$\sum_{j=1}^{m} \sum_{t=1}^{f} A_{jt} x_{ijt} = 1 \qquad \text{for } i = 1, 2, \ldots, n \tag{5.13}$$

$$C_i = \sum_{j=1}^{m} \sum_{t=1}^{f} (t + p_{ij}) x_{ijt} \qquad \text{for } i = 1, 2, \ldots, n \tag{5.14}$$

$$\sum_{i=1}^{n} \sum_{s=t}^{t+p_{ij}-1} x_{ijs} \leq 1 \qquad \text{for } j = 1, 2, \ldots, m \qquad \text{for } t = 1, 2, \ldots, f \tag{5.15}$$

$$\sum_{i=1}^{n} \sum_{t=1}^{f} p_{ij} x_{ijt} + \sum_{t=1}^{f} (1 - A_{jt}) \leq y \qquad \text{for } j = 1, 2, \ldots, m \tag{5.16}$$

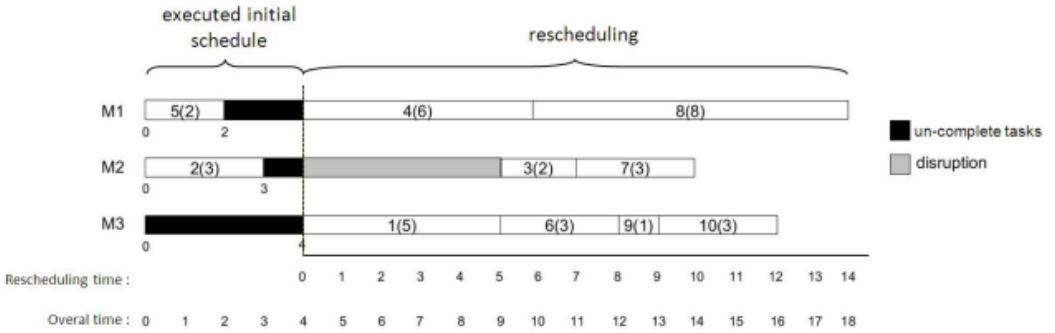

Figure 5.8: Overall executed initial schedule and rescheduling start time when $t_s = t_d$

$$D_{ij} + \sum_{t=1}^{f} x_{ijt} \geq \phi_{ij} \qquad \text{for } i = 1, 2, \ldots, n \qquad \text{for } j = 1, 2, \ldots, m \tag{5.17}$$

$$y, C_i, D_{ij} \geq 0 \quad \text{for} \quad i = 1, 2, \ldots, n \qquad j = 1, 2, \ldots, m \tag{5.18}$$

$$x_{ijt} \in \{0, 1\} \qquad \text{for} \quad i = 1, 2, \ldots, n \qquad j = 1, 2, \ldots, m \qquad t = 1, 2, \ldots, f \tag{5.19}$$

The objective function Q is as described in equation (5.10). Constraints (5.13) give restriction to the task assignment at the disruption time slot. Constraints (5.14) define the completion time of a task. Constraints (5.15) ensure that only a task is progressing during the execution time slot. Constraints (5.16) represent the completion time on a processor is less than makespan. Constraints (5.17) represent the deviation cost for the task migration and constraints (5.18) require the non-negativity for the deviation cost and makespan. Constraints (5.19) define the 0-1 variables used in the model.

Figures 5.8 and 5.9 are the total completed schedule from the execution of initial schedule and the rescheduling phase with two different t_s. These figures are the solutions for the initial schedule that having disruption during the execution as

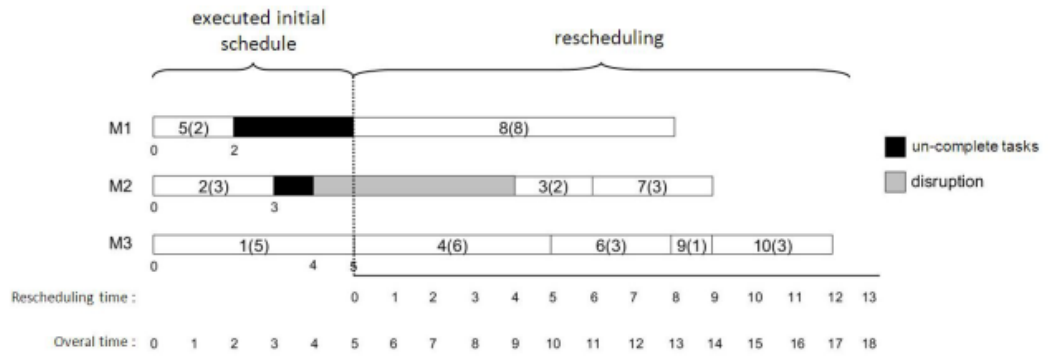

Figure 5.9: Overall executed initial schedule and rescheduling start time when $t_s = (t_d + p_d) - \delta$

in Figure 5.5. As in Figure 5.8 where the case $t_s = t_d$, the value for the y is 14 and make the overall makespan is $t_s + y = 4 + 14 = 18$. The disruption costs $z = D_{ij}$ are as follows:

$$D_{ij} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}_{8\times 3} \tag{5.20}$$

where $i = 1, 2, \ldots, 8$ and $j = 1, 2, 3$. The reset index in the rescheduling represent the task $\{J_1, J_3, J_4, J_6, J_7, J_8, J_9, J_{10}\}$. From the matrix $i \times j$, the deviation cost contributed from the migration J_4 on M_2 to another processor which is M_1 for the rescheduling phase. The total deviation cost, z, for this rescheduling is $\sum_i \sum_j D_{ij} = 1$. Hence, for $\omega_1 = \omega_2 = 0.5$, $Q = (0.5)18 + (0.5)1 = 9.5$.

For the second case where $t_s = (t_d + p_d) - \delta$ (as shown in Figure 5.9), the rescheduling makespan is 12 as only 7 tasks which are $\{J_3, J_4, J_6, J_7, J_8, J_9, J_{10}\}$ in the rescheduling system instead of $n = 8$ for the $t_s = t_d$ case. However, for the overall makespan, this example has $t_s + y = 5 + 12 = 17$ which still lower then the $t_s = t_d$ case. The disruption cost, , D_{ij} recorded as in equation (5.2.2) where the J_4 migrate from M_2. After the rescheduling, J_4 is assigned to M_3. The overall

performance of the objective function is $Q = \omega_1 y + \omega_1 z = (0.5)17 + (0.5)1 = 9$.

$$D_{ij} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}_{7\times 3} \tag{5.21}$$

5.3 Computational Experiments

In this computational experiment, we conduct a simulation for solving a disruption problem using the proposed model. In the test, we consider the post-disruption model using the rescheduling start time same as the disruption start time to reduce the delay in the rescheduling. To perform the model, firstly, we have to construct the initial schedules. The initial schedules are obtain using an optimum $R||C_{max}$ model. The variables from the initial schedules are transferred to the disruption model as parameter. Lastly, we implement the model and analyze the obtained results.

5.3.1 Computational Design

We generate the initial schedule and implement the disruption model using AIMMS 3.10 software on a PC with Intel Core 2 2.66 GHz 1.95 GB RAM. We setup a deterministic disruption environment with different length of disruption slots at the beginning of the schedule. We construct two types of data sets. The first data set for initial schedules are generated using the following simulation data:

1. In the system, there are 20 and 50 number of tasks.
2. The tasks are assigned to 2, 3 and 4 number of processors. In total, there are 6 combinations of processors and tasks.
3. For every combination, we generate 20 instances. We compute the processing time as the following uniform distribution: $p_{ij} = U[\min p_{ij}, \max p_{ij}] = U[1, 100]$

The second data set is larger than the first set and generated as follows:

1. The total number of task enter the system, $n = \{100, 200\}$
2. For every set of tasks, there are three number of processors used where $m = \{6, 8, 10\}$
3. The same distribution as in the first data set is used to computed the processing time.

The simulation data for the initial schedule will be use in the rescheduling for disruption model together with the variables obtained from the initial schedule. The following are the details to complete the rescheduling simulations for the disruption model:

1. The disruption execution time is generated as follows: $p_{i_d j_d} = L_\ell \max\{p_{ij}\}$ where $L_\ell = 1, 0.5, 0.25$ for $\ell = 1, 2, 3$.
2. Three different pair of weight for the objective function Q : $\{\omega_1, \omega_2\} = \{0.3, 0.7\}, \{0.5, 0.5\}, \{0.7, 0.3\}$

5.3.2 Computational Results

In the following, we present the result obtained by the task rescheduling for the disruption problem. We perform the experiment to evaluate the recovery MILP model of the resource disruption problem. Note that, in the testing we only consider the case where the initial schedules are optimum. There are two types of results that we evaluate. The first results are on the stability measure of the model and secondly, the model performance at different time limit.

Stability measure on different size problem Table 5.2 shows the average of the stability measure and the CPU time for the proposed recovery MILP model that has been implemented using the first type of data set. In this experiment, we deal with 6 combination of tasks and processors with 3 disruption levels, 3 weighted values and 20 replications. Therefore, we have a test data set of 1080 instances. The stability measure S, as shown in the equation (5.22), is to observe the stability condition of the current schedule compared to the initial schedule in term of the task migration which is involving disruption cost D_{ij}.

$$S = 1 - \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} D_{ij}}{n} \tag{5.22}$$

It is relevant to measure the stability of the current schedule since the initial schedule is optimum and to fulfill the goal of the proposed model (i.e maximize the match up solution of the rescheduling with the original plan). Therefore, the rescheduling model always match up with the optimum initial schedule. The most stable model is when the value of the stability measure is one where the rescheduling match up 100% with the initial schedule.

From the Table 5.2, we have conduct an experiment for the disruption problem with different levels of disruption slot at different combination of weights in the objective function. For the disruption level, L_1 is the largest disruption slot occurred in the system which is the maximum amount of the processing time in the system. Then, the L_2 and L_3 have a smaller slot with 50% and 25% less than L_1 respectively. Overall, the average of the stability measure for all specified size problem have minimum rate at 0.7 including the the case with the lowest weight consider for the total deviation cost (i.e $\omega_2 = 0.3$). The reschedule model is stable with higher stability rate when the size of problem increases. This is proved by the figure where the stability measure is higher when we compare for $n = 20$ and $n = 50$ at each disruption level. For example, we take a closer look for problem size of 20×2 and 50×2 with disruption level of L_1. The stability measure records an increment from

Table 5.2: The average of stability measure and the computational time for different combination of ω_1 and ω_2

Number of Processor m	Number of Tasks n	Disruption Level	$\omega_1 = 0.3$,$\omega_2 = 0.7$		$\omega_1 = 0.5$,$\omega_2 = 0.5$		$\omega_1 = 0.7$,$\omega_2 = 0.3$	
			Stability Measure, S	CPU (in seconds)	Stability Measure, S	CPU (in seconds)	Stability Measure, S	CPU (in seconds)
2	20	L_1	0.9075	9.4	0.8825	8.4	0.8825	19.1
		L_2	0.9375	11.0	0.9150	29.4	0.9075	358.7
		L_3	0.9400	7.9	0.9275	6.05	0.9275	2.35
	50	L_1	0.9630	203.6	0.9600	513.9	0.9580	428.5
		L_2	0.9600	556.9	0.9600	142.4	0.9520	1048.5
		L_3	0.9620	1131.7	0.9550	718.4	0.9520	596.7
3	20	L_1	0.8525	34.8	0.8300	284.45	0.8225	88.7
		L_2	0.8850	152.9	0.8525	509.35	0.8300	450.3
		L_3	0.9125	89.3	0.8625	386.95	0.8350	700.6
	50	L_1	0.9450	1054.9	0.9300	2025.7	0.9070	667.1
		L_2	0.9500	879.6	0.9340	1320.9	0.9280	1461.8
		L_3	0.9590	1197.9	0.9510	1878.3	0.9300	1107.9
5	20	L_1	0.8200	406.1	0.7975	266	0.7675	304
		L_2	0.8800	36.5	0.8600	28.25	0.8350	1527.9
		L_3	0.9175	11.1	0.8850	49.85	0.8700	161.75
	50	L_1	0.9310	1414.1	0.9310	1501.2	0.8980	2282.7
		L_2	0.9490	1610.1	0.9460	2311.5	0.9250	1475.0
		L_3	0.9540	639.13	0.9400	1616.8	0.9290	2170.9

0.9075 to 0.9630 respectively. These excellent results are consistent for all other levels and processors.

The average of CPU time record in the table are in seconds. The time is fluctuate for every level in the same size problem as there are different finish time for every instances depend on the makespan of the initial schedule. The CPU time obtain by the case of $n = 50$ is longer than $n = 20$. These occurs in the operations when a larger number enters the system and increase the usage of the operational memory. However, from the table, the proposed MILP model performed very well with a reasonable amount of time as the average of the CPU time is compute between 2.35 seconds and 38.5 minutes until the optimum solutions.

Gap on different time limit of computation time We now present the performance of the disruption model that use the second data set with $\omega_1 = \omega_2 = 0.5$ and 10 simulation problems for each test case. In this experiment, the gap is recorded at the specified solving time limit that obtained from the following equation:

$$Gap(\%) = \frac{\text{Best solution - Best LP bound}}{\text{Best solution}} \times 100 \tag{5.23}$$

where the best LP bound is refers to the lower bound obtain from the model.

The motivation of the experiment is to observe the quality of the solving time of the model for a larger data set. The MILP model is tested with 9 different stopping limits (in seconds): 30, 60, 150, 300, 600, 900, 1200, 1500, 1800.

Table 5.3 shows the average of the gap $(\%)$ recorded at different time limit. From this experiment, we can see the performance of the model from the beginning of 30 second until the last stopping limit at 30 minutes. At time $t = 30$, all the instances have the gap less than 30% from the optimum and at $t = 300$ all cases already less than 10% gap. In other words, the model have a good quality of solutions

Processor m	Task n	Level L	30	60	150	300	600	900	1200	1500	1800
6	100	L_1	6.07	5.10	4.09	3.76	2.82	2.74	2.46	2.42	2.35
		L_2	6.23	4.62	3.88	3.51	2.57	2.42	2.29	2.18	2.07
		L_3	5.76	4.04	3.02	2.69	2.22	2.11	2.07	2.01	2.00
	200	L_1	4.37	3.82	2.99	2.62	2.31	2.13	1.82	1.78	1.56
		L_2	4.68	3.65	2.46	2.31	2.21	1.94	1.87	1.65	1.63
		L_3	3.72	2.70	1.83	1.52	1.43	1.25	1.20	1.13	1.11
8	100	L_1	9.15	8.00	6.60	6.12	4.98	4.61	4.56	4.47	4.14
		L_2	12.93	9.26	7.22	6.45	4.85	4.55	4.39	3.99	3.98
		L_3	8.36	5.62	5.06	4.15	3.37	3.32	3.26	3.13	3.08
	200	L_1	6.73	4.92	4.16	3.63	3.60	3.48	3.47	3.30	3.18
		L_2	8.46	5.50	4.19	3.90	3.70	3.54	3.38	3.33	3.18
		L_3	6.82	4.66	3.68	3.41	3.23	2.69	2.42	2.39	2.36
10	100	L_1	11.66	9.92	8.08	7.59	6.41	5.70	5.55	5.55	5.55
		L_2	15.97	12.02	8.84	7.99	7.24	7.17	7.16	7.08	6.99
		L_3	12.05	7.29	6.41	6.08	5.85	5.74	5.64	5.43	5.23
	200	L_1	25.76	15.61	9.41	5.57	5.33	5.23	5.22	5.14	5.12
		L_2	22.55	15.29	11.31	7.16	5.88	5.85	5.71	5.34	4.73
		L_3	16.36	9.98	7.81	5.12	4.47	4.12	3.99	3.97	3.92

Table 5.3: Average gap at the specified stopping criteria

and can be obtained within a short amount of computational time.

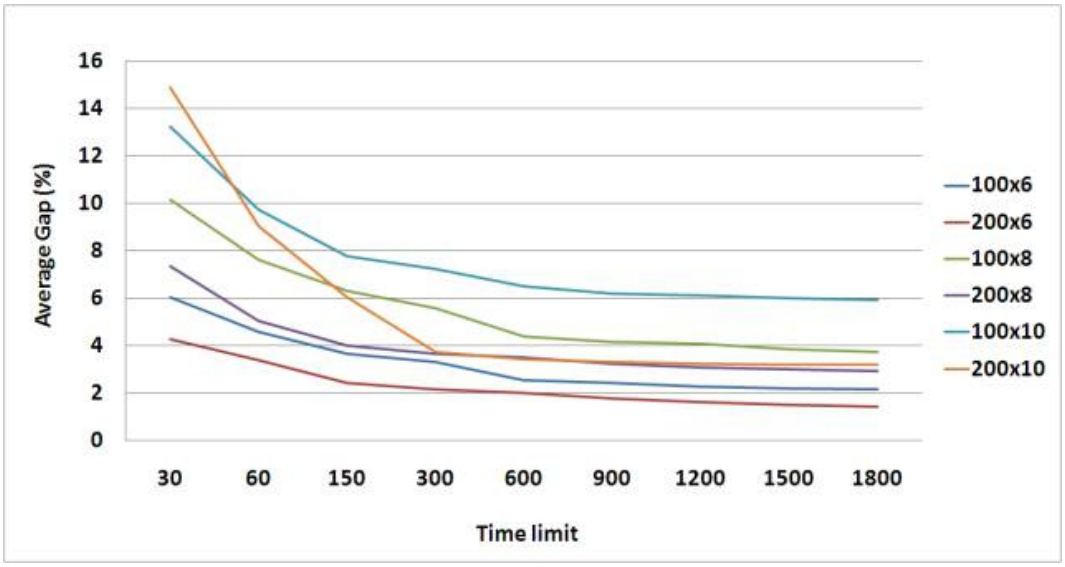

Figure 5.10: Average gap for all three disruption levels

Figure 5.10 shows the average gap for all three disruption levels at respective combination of tasks and processors that we described as $n \times m$. In the figure, the $x - axis$ indicate the percentage of the average gap while the $y - axis$ is the time limit (in seconds) that we consider. The graph is obtained from the previous table and illustrates the performance of the model for every combination of tasks and processors at the beginning of the computational time. In the diagram, we can see that the performance of the system with 200 tasks are better than 100 tasks for every processor at the first 30 minutes. The most interesting observation is when the largest data set (i.e 200×10) improve very fast within 300 seconds of the CPU times. It indicates the model can obtained small gap for a larger data set before reach the optimum value in short computational time.

5.4 Summary

In this chapter, we introduced a task scheduling problem on unrelated parallel processor system having a disruption that deals with availability of the resources. A recovery option for the disruption problem is to reschedule the initial plan and con-

sider the new disruption environment. In order to revise the initial schedule, we introduce a MILP model for predictive disruption and post-disruption case. In predictive disruption, a recovery MILP model with dummy insertion is proposed. We developed a MILP model for the post-disruption case with recovery objectives and recovery constraints as a match up strategy to stay close and minimize the deviation with the initial schedule. The MILP model for the post-disruption problem has been implemented with two different data sets to examine the stability measure and the performance of the gap at different computational time. The model performed with high value of the stability measure for all tested cases. The most obvious outstanding performance is the stability measure which is improved when the number of the tasks increase and obtain optimal result within a reasonable amount of time. For a larger data set, the model is very effective and has obtained a low gap that is less than 7.99% within 300 seconds of the elapsed time.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Task scheduling on parallel processor systems is an area with very interesting and challenging problems. The process of the task scheduling is important in order to arrange all the incoming tasks in the system and accommodate it suitable with specified operation requirements. In addition, the task scheduling system may have their own task characteristics and processor environment that needs to be considered at the same time. Furthermore, in real practise, the task scheduling problem has many applications that have been widely used in various industries. Our aim is to enhance the classical task scheduling problem with additional features that are applicable to the current situation. Specifically, in this thesis, we have developed algorithms in solving task scheduling with several different features which are extended from classical identical and unrelated parallel processors problems.

The first task characteristic that we considered is the on-line scheduling problem on identical processors where the information of the processing time and the release date of the task is non-deterministic. On-line scheduling is considered to handle the dynamic task scheduling characteristic that exist in the parallel processor system. We developed three heuristic algorithms to solve the problem since there are no optimal solutions for the on-line scheduling problem. Furthermore, we try to propose fast and efficient algorithms that are easy to implement.

In particular, the heuristics have two important procedures which are the task and processor selection schemes. The task selection offered is a multi-stage procedure in priority rule loop. The stages contain Cluster Insertion and Local Cluster Interchange methods. In the processor selection process, a greedy algorithm is performed. The heuristics are able to achieve less than 6.13% average gap from the optimal solution. The best heuristic performed very well with the maximum average gap of 1.099%. All the heuristics were also very efficient even when the size of the problem are increasing.

A new additional task characteristic which is priority consideration, has been presented for the second problem. The priority consideration is a feature for a set of tasks that have been listed in a priority list. In this feature, a task may start earlier than another task in front of the list if both the task are not assigned on a same processor. This characteristic gives way to other tasks getting assigned earlier on other processors without increasing the value of objective function. Therefore, the characteristic is offered to unrelated parallel processor environment since the processing time of a task is different on every processor.

For the problem, mixed integer programming models are developed which consider three different types of task priority consideration characteristics: ascending, descending and general priority list. The model obtained optimal solutions for all tested instances. Different sizes of problem have been tested to evaluate the performance of the computational time. On average, the model can be solved optimally less than 39.6 minutes with up to 100 jobs and 5 machines. The model is very efficient with 90.9% of the tested instances have been solved at time less than 150 seconds.

In the classical problem, the status of the processor in the system always available for processing without interruption. However, in the real world application, the

changes are always happen in the operation system due to the disruption. Therefore, we focus on another interesting features which is task scheduling problem on parallel processor system having a resource disruption for predictive disruption and post-disruption cases. The resource disruption might occurs when there is a processor become unavailable to operate. Due to this, it is necessary to revise the scheduling to prevent the infeasibility.

Our strategy in solving the disruption problem is rescheduling the initial schedule. The rescheduling approach is a recovery option to manage the disruption by address new constraints and new objectives. Mixed integer linear programming models are formulated for the recovery model with dummy insertion and match-up schedule technique. A computational study is conducted to evaluate the stability of the rescheduling model in term of the deviation from the original schedule. The experiments indicate that the stability values are getting better when the size of the problem increases. The average gap for a larger data set is good since the first 30 second of the computational time. Then, the gaps are reducing until obtain less than 7.99% starting at 300 second of CPU time.

6.2 Future Work

Task scheduling problem on unrelated parallel processor has considerable potential for further study. One problem that can be considered is to improve the proposed local search algorithm for the multi-stage scheduling for on-line problem. Every single step could have a wider exploration strategy towards the dynamic arrival tasks in the system. Meta-heuristic mechanisms could be developed to overcome the local optimum trap. Another study that could be considered is an improvement on the models with priority consideration and disruption problem in order to reduce the computational time in obtains optimal result for a large data set. The nearly optimum heuristics also can be developed for the models in order to improve the

time.

It would be interesting if the implementation have more added components in the computational design. For example, computational testing on different ranges in the uniform distribution of the processing time. In the disruption problem, different types of the initial schedule can be proposed to observed the effect of the initial schedule on the stability of the rescheduling model.

Multiple performance measures and characteristics could be considered in future which have many applications in the practice. The features that have been described in this thesis could be applied to other processor environments as well. The experiments that used the real scenario data can also be significant for the extensive study.

The disruption problem in scheduling can be extend using other interruption constraints for example task disruptions, changing in orders from customers and any external factors that effect the scheduling. Online disruption problem also can be solved in future. It is a very valuable and relevant problem in real applications.

Bibliography

Akyol, D. E. and Bayhan, G. M. (2007). A review on evolution of production scheduling with neural network. *Computers and Industrial Engineering*. 53: 95–122.

Alagoz, O. and Azizoglu, M. (2003). Rescheduling of identical parallel machines under machine eligibility constraints. *European Journal of Operational Research*. 149: 523–532.

Albers, S. (2003). On-line algorithms : a survey. *Mathematical Programming*. 97: 3–26.

Albers, S. and Fujiwara, H. (2007). Energy-efficient algorithm for flow time minimization. *ACM Transaction on Algorithms*. 3: 49:1–49:17.

Allahverdi, A. and Al-Anzi, F. S. (2008). The two-stage assembly flowshop scheduling problem with bicriteria of makespan and mean completion time. *International Journal of Advanced Manufacturing*. 37: 166–177.

Alvim, A. C. and Ribeiro, C. C. (2004). A hybrid bin-packing heuristic to multiprocessor scheduling. *Lecture Notes in Computer Science*. 3059: 1–13.

Amico, M. D., Iori, M., Martello, S. and Monaci, M. (2008). Heuristic and exact algorithms for the identical parallel machine scheduling problem. *INFORMS Journal on Computing*. 20: 333–344.

Andersson, B. (2003). *Static-priority scheduling on multiprocessors*. Sweden: Ph.D thesis, Chalmers University of Technology in Goteborg.

Arnaout, J.-P. and Rabadi, G. (2008). Rescheduling of unrelated parallle machines under machine breakdowns. *International Journal of Applied Management Science*. 1: 75–89.

Aytug, H., Lawley, M. A., Kenneth McKay, S. M. and Uzroy, R. (2005). Executing production schedules in the face of uncertainties: a review and some future directions. *European Journal of Operational Research*. 161: 86–110.

Azizoglu, M. and Alagoz, O. (2005). Parallel-machine rescheduling with machine disruptions. *IIE Transactions*. 37: 1113–1118.

Baccelli, F., Liu, Z. and Towsley, D. (1993). Extremal scheduling of parallel processing with and without real-time constraints. *Journal of Association for Computing Machinery*. 40: 1209–1237.

Bekki, O. B. and Azizoglu, M. (2008). Operational fixed interval scheduling problem on uniform parallel machines. *International Journal of Production Economics*. 112: 756–768.

Bilge, U., Kiras, F., Kurtulan, M. and Pekgun, P. (2004). A tabu search algorithm for parallel machine total tardiness problem. *Computer and Operational Research*. 31: 397–414.

Blazewicz, J., Dror, M. and Weglarz, J. (1991). Mathematical programming formulations for machine scheduling : a survey. *European Journal of Operational Research*. 51: 283–300.

Blazewicz, J. and Kobler, D. (2002). Review of properties of different precedence graphs for scheduling problems. *European Journal of Operational Research*. 142: 435–443.

Caccetta, L. and Nordin, S. Z. (2010). MILP model in minimizing makespan with priority for unrelated parallel machines. In *Proceedings of the International Conference of Optimal Control and Optimization*. Vol. 4. Guiyang China. 42–46.

Chandha, J. S., Garg, N., Kumar, A. and Muralidhara, V. (2009). A competitive algorithm for minimizing weighted flow time on unrelated machines with speed augmentation. In *Proceeding of the 41st Annual ACM Symposium on Theory of Computing 2009 (STOC '09)*. Maryland, USA. 679–684.

Chaudhry, I. A. (2010). Minimizing flow time for the worker assignment problem in identical parallel machine model using GA. *International Journal of Manufacturing Technology*. 48: 747–760.

Chen, B., Potts, C. N. and Woeginger, G. J. (1998). *A review of machine scheduling: complexity, algorithms and approximability in "Handbook of Combinatorial Optimization (Vol. 3)" (eds. D.-Z. Du and P.M. Pardalos)*. Kluwer Academic Publishers.

Chen, C.-L. and Chen, C.-L. (2009a). A bottleneck-based heuristic for minimizing makespan in a fleible flow line with unrelated parallel machines. *Computers and Operations Research*. 36: 3073–3081.

Chen, C.-L. and Chen, C.-L. (2009b). Scheduling of unrelated parallel machines: an application to PWB manufacturing. *Computer and Operations Research*. 36: 3073–3081.

Cheng, T. C. E. and Sin, C. C. S. (1990). A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*. 47: 271–292.

Ciavotta, M., Meloni, C. and Pranzo, M. (2009). Scheduling dispensing and counting in secondary pharmaceutical manufacturing. *AIChe Journal*. 55: 1161–1170.

Clausen, J., Larsen, A., Larsen, J. and Rezanova, N. J. (2010). Disruption management in the airline industry - concepts, model and methods. *Computers and Operations Research*. 37: 809–821.

Cosnard, M. and Trystram, D. (1995). *Parallel algorithms and architectures*. London: International Thomson Publishing.

Davis, E. and Jaffe, J. (1981). Algorithm for scheduling tasks on unrelated processors. *Journal of Association for Computing Machinery*. 28: 721–736.

De, P. and Morton, T. (1980). Scheduling to minimize makespaan on unequal parallel processors. *Decision Sciences*. 11: 586–603.

Dhingra, A. and Chandna, P. (2010). Multi-objective flow shop scheduling using hybrid simulated annealing. *Measuring Business Excellence*. 14: 30–41.

Doganis, P. and Sarimveis, H. (2008). Optimal production scheduling for the dairy industry. *Annals of Operations Research*. 159: 315–331.

Dorndorf, U., Jaehn, F., Lin, C., Ma, H. and Pesch, E. (2007). Disruption management in flight gate scheduling. *Statistica Neerlandica*. 61: 92–114.

Du, D.-Z. and Pardalos, P. (1998). *Handbook of Combinatorial Optimization (Vol.3)*. Kluwer Academic Publishers.

Du, J. and Leung, J. Y.-T. (1989). Complexity of scheduling parallel task systems. *SIAM Journal on Discrete Mathematics*. 2: 473–487.

El-Rewini, H., Lewis, T. G. and Ali, H. H. (1994). *Task scheduling in parallel and distributed systems*. New Jersey: Prentice Hall.

Epstein, L. (2000). A note on-line scheduling with precedence constaraints on identical machines. *Infromation Processing Letters*. 76: 149–153.

Fanjul-Peyro, L. and Ruiz, R. (2010). Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*. 207: 55–69.

Frangioni, A., Necciari, E. and Scutella, M. G. (2004). A multi-exchange neighborhood for minimum makespan parallel machine scheduling problems. *Journal of Combinatioral Optimization*. 8: 195–220.

Fu, R., Tian, J., Yuan, J. and He, C. (2008). On-line scheduling on a batch machine to minimize makespan with limited restarts. *Operations Research Letters*. 36: 255–258.

Fujimoto, N. and Hagihara, K. (2006). A 2-approximation algorithm for scheduling independent task onto a uniform parallel machine and its extension to a computational grid. In *IEEE International Conference on Cluster Computing*. 1–7.

Funk, S., Goossens, J. and Baruah, S. (2001). On-line scheduling on uniform multiprocessor. In *22nd IEEE Procedding of Real-time Systems Symposium*. 183–192.

Gao, W.-J., Huang, X. and Wang, J.-B. (2010). Single-machine scheduling with precedence constraints and decreasing start-time dependent processing time. *The International Journal of Advanced Manufacturing Technology*. 46: 291–299.

Ghirardi, M. and Potts, C. (2005). Makespan minimization for scheduling unrelated parallel machines: a recovering beam search approach. *European Journal of Operational Research*. 165: 457–467.

Glass, C., Potts, C. and Shade, P. (1994). Unrelated parallel machine scheduling using local search. *Mathematical and Computer Modelling*. 20: 41–52.

Gordon, V., Proth, J.-M. and Chu, C. (2002a). A survey of the state-of-the-art of common dua date assignment and scheduling research. *European Journal of Operational Reasearch*. 139: 1–25.

Gordon, V., Proth, J.-M. and Chu, C. (2002b). A survey of the state-of-the-art of common due date assignment and scheduling research. *European Journal of Operational Research*. 139: 1–25.

Goren, S. and Sabuncuoglu, I. (2010). Optimization of schedule robustness and stability under random machien breakdown and processing time variability. *IIE Transactions*. 42: 203–220.

Graham, R. (1969). Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*. 17: 416–429.

Graham, R. L., Lawler, E. L., Lenstra, J. K. and Kan, A. H. G. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*. 5: 287–326.

Gualtieri, M. I., Paletta, G. and Pietramala, P. (2008). A new $n\ log\ n$ algorithm for the identical parallel machine scheduling problem. *Int. J. Contemp. Math. Sciences*. 13: 25–36.

Hall, N. G., Liu, Z. and Potts, C. N. (2007). Rescheduling for multiple new orders. *INFORMS Journal on Computing*. 19: 633–645.

Hall, N. G. and Potts, C. N. (2004). Rescheduling for new orders. *Operations Research*. 52: 440–453.

Haouari, M. and Gharbi, A. (2004). Lower bounds for scheduling on identical parallel machines with heads and tails. *Annals of Operation Research*. 129: 187–204.

Hartmann, W., Fischer, A. and Nyhuis, P. (2004). The impact of priority rules on logistic objectives: modelling with the logistic operating curve. In *Proceedings of the world congress on engineering and computer science*. Vol. II.

Haupt, R. (1989). A survey of priority rule-based scheduling. *OR Spektrum*. 11: 3–16.

Hochbaum, D. S. and Shmoys, D. B. (1987). Using dual approximation algorithms for scheduling problem: theoretical and practical results. *Journal of the Association for Computing Machinery*. 34: 144–162.

Horowitz, E. and Sahni, S. (1976). Exact and approximate algorithms for scheduling nonidentical processor. *Journal of Association for Computing Machinery*. 23: 317–327.

Huang, Y.-M. and Lin, J.-C. (2011). A new bee colony optimization algorithm with idle-time-based filtering scheme for open shop-scheduling problem. *Expert Systems with Applications*. 38: 5438–5447.

Huo, Y., Leuong, J. Y. T. and Wang, X. (2008). On-line scheduling of equal-processing time task systems. *Theoretical Computer Science*. 401: 85–95.

Hurink, J. and Paulus, J. (2008). Online scheduling of parallel jobs on two machines is 2-competitive. *Operations Research Letters*. 36: 51–56.

Ibarra, O. and Kim, C. (1978). Heuristic algorithms for scheduling independent task on nonidentical processors. *Journal of Association for Computing Machinery*. 25: 612–619.

Imai, A., Nishimura, E. and Papadimitriou, S. (2003). Berth allocation with service priority. *Transportation Reserach Part B*. 37: 437–457.

Ji, M. and Cheng, T. (2008). Parallel-machine scheduling with simple linear deterioration to minimize total completion time. *European Journal of Operational Research*. 188: 342–347.

Jinsong, B., Xiaofeng, H. and Ye, J. (2009). A genetic algorithm for minimizing makespan of block erection in shipbuilding. *Journal of Manufacturing Technology Management*. 20: 500–512.

Kachitvichyanukul, V. and Sitthitham, S. (2011). A two-stage genetic algorithm for multi-objective job shop scheduling problems. *Journal of Intelligent Manufacturing*. 22: 355–365.

Kellerer, H. (2008). An approximation algorithm for identical parallel machine scheduling with resource dependent processing times. *Operations Research Letters*. 36: 157–159.

Kim, D.-W., Na, D.-G. and Chen, F. F. (2003). Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective. *Robotics and Computer Integrated Manufacturing*. 19: 173–181.

Klein, R. (2000). Bidirectional planning: Improving priority rule-based heuristic for scheduling resource-constarint projects. *European Journal of Operations Research*. 127: 619–638.

Kohl, N., Larsen, A., Larsen, J., Ross, A. and Tiourine, S. (2007). Airline disruption management - perspectives, experiences and outlook. *Journal of Air Transport Management*. 13: 149–162.

Kolen, A. W. and Kroon, L. G. (1991). On the computational complexity of (maximum) class scheduling. *European Journal of Operational Research*. 54: 23–38.

Koulamas, C. (2011). A unified solution approach for the due date assignment problem with tardy jobs. *International Journal of Production Economics*. 132: 292–295.

Koulamas, C. and Kyparisis, G. J. (2000). Scheduling on uniform parallel machines to minimize maximum lateness. *Operations Research Letters*. 26: 175–179.

Koulamas, C. and Kyparisis, G. J. (2009). A modified LPT algorithm for the two uniform parallel machine makespan minimization problem. *European Juornal of Operational Research*. 196: 61–68.

Kravchenko, S. A. and Werner, F. (2009). Preemtive scheduling on uniform machine to minimize mean flow time. *Computers and Operations Research*. 36: 2816–2821.

Kumar, V. S. A., Marathe, M. V., Parthasarathy, S. and Srinivasan, A. (2009). Scheduling on unelated machines under tree-like precedence constraints. *Algorithmica*. 55: 205–226.

Lam, T.-W., Lee, L.-K., To, I. K. and Wong, P. W. (2008). Speed scaling functions for flow time scheduling based on active job count. *Lecture Notes in Computer Science*. 5193: 647–659.

Lauff, V. and Werner, F. (2004). Scheduling with common due date, earliness and tardiness penalties for multimachine problems: a survey. *Mathematical and Computer Modelling*. 40: 637–655.

Lawler, E. and Labetoulle, J. (1978). On preemtive scheduling of unrelated parallel processors by linear programming. *Journal of the Association for Computing Machinery*. 25: 612–619.

Lee, C.-Y. and Yu, G. (2007). Single machine scheduling under potential disruption. *Operations Research Letters*. 35: 541–548.

Lee, K., Leung, J. Y. T. and Pinedo, M. L. (2010). On-line scheduling with machine eligibility. *A Quarterly Journal of Operations Research*. 8: 331–364.

Lei, D. (2009). Multi-objective production scheduling: a survey. *International Journal of Advanced Manufacturing Technology*. 43: 926–938.

Lenstra, J. K., Shmoys, D. B. and Tardos, E. (1990). Approximation algorithms for scheduling unrelated parallel machine. *Mathematical Programming*. 46: 259–271.

Lenstra, J. and Kan, A. R. (1978). Complexity of scheduling under precedence constraint. *Operation Research*. 26: 22–35.

Li, K. (2005). Job scheduling and processor allocation for grid computing and metacomputers. *Journal of Parallel and Distributed Computing*. 65: 1406–1418.

Li, K. and Yang, S.-L. (2009). Non-identical parallel-machine scheduling research with minimizing total weighted completion times: model, relaxations and algorithms. *Applied mathematical Modelling*. 33: 2145–2158.

Li, K., Yang, S.-L. and Ma, H.-W. (2011). A simulated annealing approach to minimize the maximum lateness on uniform parallel machines. *Mathematical and Computer Modelling*. 53: 854–860.

Liao, C.-J. and Lin, C.-H. (2003). Makespan minimization for two uniform parallel machine. *International Journal of Production Economics*. 84: 205–213.

Lin, C.-H. and Liao, C.-J. (2008). Makespan minimization for multiple uniform machines. *Computer and Industrial Engineering*. 54: 983–992.

Liu, C. L. and Layland, J. W. (1990). Scheduling algorithms for multiprogramming in a hard-real-time environment. *IEEE Transaction on Computers*. 39: 1175–1185.

Liu, C. and Yang, S. (2011). A heuristic serial schedule algorithm for unrelated parallel machine scheduling with precedence constarints. *Journal of Software*. 6: 1146–1153.

Liu, M., Xu, Y., Chu, C. and Zheng, F. (2009). Online scheduling on two uniform machines to minimize the makespan. *Theoretical Computer Science*. 410: 2099–2109.

Liu, P. and Lu, X. (2009). On-line scheduling of parallel machine to minimize total completion time. *Computer and Operations Research*. 36: 2647–2652.

Liu, Z. (2010). Single machine scheduling to minimize maximum lateness subject to release dates and precedence constarints. *Computer & Operations Research*. 37: 1537–1543.

Lochtefeld, D. F. and Ciarallo, F. W. (2011). Helper-objective optimization strategies for the job-shop scheduling problem. *Applied Soft Computing*. 11: 4161–4174.

Low, C., Ji, M., Hsu, C.-J. and Su, C.-T. (2010). Minimizing the makespan in a single machine scheduling problems with flexible and periodic maintanance. *Applied Mathematical Modelling*. 34: 334–342.

Martello, S., Soumis, F. and Toth, P. (1997). Exact and approximation algorithms for makespan minimization on unrelated parallel machine. *Discrete Applied Mathematics*. 75: 169–188.

Mokotoff, E. (2004). An exact algorithm for the identical parallel machine scheduling problem. *European Journal of Operational Research*. 152: 758–769.

Mokotoff, E. and Chretienne, P. (2002). A cutting plane algorithm for the unrelated parallel machine scheduling problem. *European Journal of Operational Research*. 141: 515–525.

Mokotoff, E. and Jimeno, J. (2002). Heuristic based on partial enumeration for the unrelated parallel processor scheduling problem. *Annals of Operations Research*. 117: 133–150.

Moldovan, D. I. (1993). *Parallel processing: from Application to systems*. California: Morgan Kaufman Publishers.

Moratori, P., Petrovic, S. and Vazquez, A. (2008). Match-up strategies for job shop rescheduling. *New Frontiers in Applied Artificial Intelligence*. 5027: 119–128.

Naderi, B., Ghomi, S. F., Aminnayeri, M. and Zandieh, M. (2011). Scheduling open shops with parallel machines to minimize total completion time. *Journal of Computational and Applied Mathematics*. 235: 1275–1287.

Nordin, S. Z. and Caccetta, L. (2009). A heuristic to minimize makespan in dynamic task scheduling on multiprocessors. In *Proceedings of the 5th Asian Mathematical Conference(AMC2009)*. Kuala Lumpur Malaysia. 471–478.

Omara, F. A. and Arafa, M. M. (2010). Genetic algorithms for task scheduling problem. *Journal of Parallel and Distributed Computing*. 70: 13–22.

Ozlen, M. and Azizoglu, M. (2009). Generating all efficient solutions of a rescheduling problem on unrelated parallel machines. *International Journal of Production Research*. 47: 5245–5270.

Ozlen, M. and Azizoglu, M. (2011). Rescheduling unrelated parallel machines with total flow time and total disruption cost criteria. *Journal of the Operational Research Society*. 62: 152–164.

Panahi, H. and Tavakkoli-Moghaddam, R. (2011). Solving a multi-objective open shop scheduling problem by a novel hybrid ant colony optimization. *Expert Systems with Applications*. 38: 2817–2822.

Parsa, N. R., Karimi, B. and Kashan, A. H. (2010). A branch and price algorithm to minimize makespan on single batch processing machine with non-identical job sizes. *Computer and Operations Research*. 37: 1720–1730.

Potts, C. (1985). Analysis of a linear porgramming heuristic for scheduling unrelated parallel machines. *Discrete Applied Mathematics*. 10: 155–164.

Qi, X., Bard, J. F. and Yu, G. (2006). Disruption management for machine scheduling: the case of SPT schedules. *International Journal of Production Economics*. 103: 166–184.

Queyranne, M. and Schulz, A. S. (2006). Approximation bounds for a general class of precedence constrained parallle machine scheduling problems. *SIAM Journal on Computing*. 35: 1241–1253.

Ramachandra, G. and Elmaghraby, S. E. (2006). Sequencing precedence-related jobs on two machines to minimize the weighted completion time. *International Journal of Production Economics*. 100: 44–58.

Rangsaritratsamee, R., Jr., W. G. F. and Kurz, M. B. (2004). Dynamic rescheduling that simultaneously considers efficiency and stability. *Computers & Industrial Engineering*. 46: 1–15.

Rocha, P. L., Ravetti, M. G., Mateus, G. R. and Pardalos, P. M. (2008). Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine dependent setup times. *Computers and Operations Research*. 35: 1250–1264.

Ruiz, R. and Vazquez-Rodriguez, J. A. (2010). The hybrid flow shop scheduling problem. *European Journal of Operational Research*. 205: 1–18.

Sgall, J. (1998). *On-line scheduling*. Berlin: Springer.

Sha, L., Rajkumar, R. and Lehoczky, J. P. (1990). Priority inheritance protocols: an approach to real-time synchronization. *IEEE Transaction on Computers*. 39: 1175–1185.

Sinnen, O. (2007). *Task scheduling for parallel systems*. New Jersey: John Wiley & Sons Inc.

Srivastava, B. (1998). An effective heuristic for minimizing makespan on unrelated parallel processor. *Journal of the Operational Research Society*. 49: 886–894.

Sun, Y., Zhang, C., Gao, L. and Wang, X. (2011). Multi-objective optimization algorithms for flow shop scheduling problem: a review and prospects. *International Journal of Advanced Manufacturing Technology*. 55: 723–739.

Tang, L. and Zhang, Y. (2009). Parallel Machine Scheduling Under the Disruption of Machine Breakdown. *Industrial and Engineering Chemistry Research*. 48: 6660–6667.

Tao, J., Chao, Z. and Xi, Y. (2010). A semi-online algorithm and its competitive analysis for a single machine scheduling problem with bounded processing times. *Journal of Industrial and Management Optimization*. 6: 269–282.

Tavakkoli-Moghaddam, R. and Rahimi-Vahed, A. (2007). A hybrid multi-objective immune algorithm for a flow shop scheduling problem with bi-objective: weighted mean completion time and weighted mean tardiness. *Information Sciences*. 177: 5072–5090.

Tian, J., Fu, R. and Yuan, J. (2009). A best online algorithm for scheduling on two parallel batch machines. *Theoretical Computer Science*. 410: 2291–2294.

Toktas, B., Azizoglu, M. and Koksalan, S. K. (2004). Two-machine flow shop scheduling with two criteris: maximum earliness and makespan. *European Journal of Operational Research*. 157: 286–295.

Tuong, N. H., Soukhal, A. and Billaut, J.-C. (2009). A new dynamic programming formulation for scheduling independent tasks with common due date on parallel machines. *European Journal of Operational Research*. 202: 646–653.

Vairaktarakis, G. L. and Cai, X. (2003). The value of processing flexibility in multipurpose machines. *IIE Transactions*. 35: 763–774.

Vallada, E. and Ruiz, R. (2011). Genetic algorithms for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*. 211: 6112–622.

Vieira, G. E., Herrmann, J. W. and Lin, E. (2003). Rescheduling manufacturing system: a framework of strategies, policies, and methods. *Journal of Scheduling*. 6: 39–62.

Wan, C.-S., Shih, W.-K. and Chang, R.-C. (2003). Real-time packet scheduling in next generation radio access system. *Computer Communications*. 26: 1931–1943.

Wang, X., Gao, L., Zhang, C. and Shao, X. (2010). A multi-objective genetic algorithm based on immune and entropy principle for flexible job-shop scheduling problem. *International Journal of Advanced Manufacturing Technology*. 51: 757–767.

Wigderson, A. (2006). P, NP and Mathematics - A computational complexity perspective. In *Proceedings of the International Congress of Mathematicians*. Vol. 1. Madrid. 665–712.

Williams, S. K. and Magazine, M. J. (2007). Heuristic approaches for batching jobs in printed circuit board assembly. *Computers and Operation Research*. 34: 1943–1962.

Wotzlaw, A. (2007). *Scheduling unrelated paralllel machines: algorithms, complexity and performance*. Saarbrucken, Germany: VDM Verlag Dr. Muller.

Wu, Y. and Ji, P. (2009). A scheduling problem for PCB assembly: a case with multiple line. *International Journal Advanced Manufacturing Technology*. 43: 1189–1201.

Xhafa, F. and Abraham, A. (2008). *Metaheuristics for scheduling in industrial and manufacturing applications*. Berlin: Springer-Verlag.

Xiaoguang, Y. (2000). A class of generalized multiprocessor scheduling problems. *Systems Science and Mathematical Sciences*. 13: 385–390.

Xiuli, F., Wanghui, Chengxiang, Z. and Lin, Z. (2010). Optimization of job-shop scheduling using fuzzy heuristic algorithm for dicrete manufacturing exterprise. In *2010 International Conference on Computing, Comtrol and Industrial Engineering*. Vol. 2. 182–185.

Xu, D., Sun, K. and Li, H. (2008). Parallel machine scheduling with almost periodic maintenance and non-preemtive jobs to minimize makespan. *Computers and Operations Research*. 35: 1344–1349.

Xu, J. and Parnas, D. L. (2000). Priority scheduling versus pre-run-time scheduling. *The International Journal of Time-Critical Computing Systems*. 18: 7–23.

Yang, T. (2009). An evolutionary simulation-optimization approach in solving parallel-machine scheduling problems - a case study. *Computers & Industrial Engineering*. 56: 1126–1136.

Yu, G. and Qi, X. (2004). *Disruption management: framework, models and applications*. World Scientific.

Yu, L., Shih, H. M., Pfund, M., Carlyle, W. M. and Fowler, J. W. (2002). Scheduling of unrelated parallel machines: an application to PWB manufacturing. *IIE Transactions*. 34: 921–931.

Zafeiris, V. E. and Giakoumakis, E. (2008). Towards flow scheduling optimization in multihomed mobile host. In *IEEE 19th International Symposium of Personal, Indoor and Mobile Radio Communication (PIMRC)*. 1–5.

Zaidi, M., Jarboui, B., Kacem, I. and Loukil, T. (2010). Hybrid mete-heuristics for minimizing the total weighted completion time on uniform parallel machines. *Electronic Notes in Discrete Mathematics*. 36: 543–550.

Zhao, C.-L. and Tang, H.-Y. (2010). Single machine scheduling with general job-dependent aging effect and maintanance activities to minimize makespan. *Applied Mathematical Modelling*. 34: 837–841.

Zhu, G., Bard, J. and Yu, G. (2005). Disruption management for resource - constrained project scheduling. *The Journal of Operational Research Society*. 56: 365–381.