Towards Power of Preemption on Parallel Machines

Babak Takand

A thesis submitted in partial fulfilment of the requirements of the
University of Greenwich for the degree of Master of Philosophy

June 2016

Department of Mathematics,
Faculty of Architecture, Computing, and Humanities,
University of Greenwich,
London, U.K.

To my family.

DECLARATION

I certify that this work has not been accepted in substance for any degree, and is not concurrently being submitted for any degree other than that of Master of Philosophy being studied at the University of Greenwich. I also declare that this work is the result of my own investigations except where otherwise identified by references and that I have not plagiarised the work of others.

Babak Takand (Student)	Prof. Vitaly Strusevich (1st Supervisor)	Dr. Alan Soper (2nd Supervisor)

ACKNOWLEDGEMENTS

First and foremost, I would like to extend my gratitude to my supervisors, Prof. Vitaly Strusevich and Dr. Alan Soper, for their guidance, support, and most importantly for constantly challenging me to improve.

I would especially like to thank my family, who have always been there for me when I needed them the most, for their relentless support and encouragement since the very beginning of my studies.

ABSTRACT

Classical scheduling models typically fall in either of two categories: those that allow interruption of the processing of jobs, and those that do not. In parallel machine environments, scheduling problems for models which allow parallel processing of jobs are typically easier to solve, in terms of computational requirements, while these models are in the majority of cases associated with an improved quality of solutions.

In this thesis, we focus on one of the notion of preemption, a foundational concept in scheduling, which defines the ability to interrupt the processing of a job and resuming it at a later time, or in the case of multiple processors, on a different machine. Preemptive scheduling is limited to the fact that every job in a preemptive schedule may not be processed by more than one machine at a time. Additionally, we consider the closely related notion of splitting jobs, where jobs can be processed at the same time by multiple processors.

We address the issue of power of preemption and power of splitting, defined as the ratio of the cost function of an optimal non-preemptive schedule over the cost function of an optimal preemptive schedule, and schedule with splitting jobs respectively. For serveral parallel machine scheduling models we provide new results, in addition to a detailed review of the best known results.

CONTENTS

"It is obvious that our industrial technology, our use of intricate machines and immense amounts of power, seem to us to be characteristic of our age. We live, we say, in an age of machines. They give us our mobility, our communications, our potential destructiveness, our comfort, and our "preoccupation with things", if indeed it is fair to charge us with that obsession. Machines are used incidentally by the biologist and the physician also in the physiological manipulations and enquiries which give us our health and long lives. And all of these complicated devices are the fruit of sciences that could never have reached their present stage of development without the highly artificial and abstract language of mathematics."
- Lyman Bryson

CHAPTER 1

Introduction

The notion of scheduling is concerned with the timely allocation of resources, so that certain tasks can be performed. Examples of scheduling problems can be found in a variety of cases, with the nature of tasks and resources depending on the domain in which the problem applies. Resources may come in the form of machines in a manufacturing plant, storage space in a warehouse, processors and memory of computing systems, communication channels, staff in organizations, whilst others resources include airport runways and cashier tills in consumer stores. In these examples, as well as other cases where scheduling problems can be found, resources are limited in nature, and are essential to the completion of processes. Processes, also referred to as tasks or jobs throughout the literature, consist of a number of operations which consume resources provided by the system. For instance, in the manufacturing plant example given earlier, a process would consist of the fabrication and assembly of an end product. The process in this case is considered complete when all stages of manufacturing have finished, hence the end product is complete. In an analogous manner, a computer program can be considered as a single process defined by a set of instructions which consume a certain amount of resources (processor time and memory), with the program being considered complete when the last instruction is performed.

Generally scheduling problems are expressed in terms of a set of jobs, and a set of machines to which jobs are to be allocated. Such an allocation of jobs to machines is the essential definition of a schedule. Schedules are considered feasible if all constraints which are defined by the problem are satisfied. Scheduling problems are associated with a particular objective which the problem ultimately aims to optimize. This objective is expressed mathematically as a function of some resource defined in a problem of interest, hence termed *objective function.* Solving a scheduling problem is achieved by providing an algorithm which outputs either an optimal schedule, or approximates the optimal solution for computationally complex problems where an exact solution would require an impractical amount of time. Algorithms consist of a series of steps, each containing some basic operations. For some input set, these steps are performed in a preordained sequence, such that an output set is produced. For scheduling problems, the output of a scheduling algorithm is a feasible schedule which has an optimal or near optimal value of the problem's objective criterion.

Interest in scheduling algorithms from an academic standpoint emerged in the 1950s, initially as a means of increasing the effectiveness of management in workshops and production lines, by lowering the cost of production and increasing throughput. About a decade later, the emergence of early operating systems for computers signified a new era in scheduling, as the very limited resources of early computers brought forward the need for fast and efficient scheduling algorithms for the management of their resources. Despite the exponential increase in processor speeds and other resources such as volatile and non-volatile memory in modern computer systems, scheduling algorithms still have a prominent role at the heart of any modern operating system. Similarly, with the rise of distributed computing scheduling algorithms for packet-switching networks, since their emergence in the early 1960s, have become essential for all digital communications. As a matter of fact, in this modern age, there is an immense number of network infrastructure devices across the globe, which utilize extremely efficient scheduling algorithms to effectively balance loads amongst communication channels. As mentioned earlier, certain scheduling problems have a great amount of complexity: The number of steps required by some algorithm which finds a feasible schedule with the optimal objective criterion value, increases so vastly in comparison to the problem size, that solving such a problem would seem not only impractical, but often infeasible by the time frame which is require to provide a solution. For such computationally hard problems, algorithms have been devised which provide a schedule which approximates the optimal objective value, or utilizes some type of methodology -a heuristic- to provide a solution, which although may not be close to optimal, is produced in a minimal amount of time, often polynomially proportionate to the problem size, and the solution is acceptable from a practical standpoint.

In this thesis we draw our attention to scheduling models on multiple machines which work in parallel. The first results for this type of problems are provided by McNaughton (1959), who considers the case in which all machines are exactly alike, i.e. identical. In the same work, McNaughton introduces the concept of preemption on parallel machines, where the processing of a job is allowed to be interrupted in order to be processed on a different machine. The performance gain of using preemption as opposed to schedules where preemption of jobs is not allowed has been vastly investigated for several decades for identical machines as well as variations of the problem where machines are non-identical, and have various processing capabilities.

For a number of classical parallel machine models, we perform an analysis of preemptive and job splitting schedules for scheduling problems with the makespan and total flow time objectives. In the next sections of this chapter, we introduce essential concepts, notation, algorithms and the theory behind them, in order to provide the theoretical framework for our findings in Chapter 5.

1.1 Machine Scheduling: Models and Notation

In this thesis we are concerned with a family of classical scheduling problems on parallel machines, in which the processing of jobs defined by the problem may take place on multiple machines. Generally, all the scheduling problem of consideration are defined by a set of jobs $N = \{J_1, J_2, \ldots, J_n\}$ which are to be processed on a set of machines $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$, with the objective of optimizing some objective function Φ. For problems of our consideration, jobs are associated with a processing time p_{ij}, such that if some job $J_j,\ 1 \le j \le n$, is assigned to a machine $M_i,\ 1 \le i \le m$, job $J_j,\ 1 \le j \le n$, would require p_{ij} amount of time. For the description of scheduling problems in this thesis, we use the representation scheme of Graham, Lawler, Lenstra and Kan (1979) for machine scheduling problems, where the three field notation $\alpha \mid \beta \mid \gamma$ is used. Under this scheme, field α represents the machine environment, field β the job characteristics and finally, field γ encapsulates the objective function of the problem. A summary of the notation used throughout this thesis is provided next:

• N	: Set of jobs $\{J_1, J_2, \ldots, J_n\}$.
• n	: Number of jobs.
• j	: Job index, $1 \le j \le n$.
• $\mathcal{M}$	: Set of machines $\{M_1, \ldots, M_m\}$.
• m	: Number of machines.
• i	: Machine index, $1 \le i \le m$.
• p_{ij}	: Processing time of job $J_j,\ 1 \le j \le n,$ on machine $M_i \in \mathcal{M}$.
• s_i	: Speed of machine M_i.
• C_j	: Completion time of a job $J_j,\ 1 \le j \le n$.

1.1.1 Machine Environments

The machine environment of a scheduling problem is defined by the value of field α in scheduling notation. This essentially provides a description of the configuration in which machines operate. Typically two classes of machine environments are observed in classical scheduling problems. Environments where each job consists of a single operation are classified as single-stage, whist machine environments in which jobs consist of multiple operations, where each operation requires a specific machine, are defined as multi-stage. Each of these machine environment classes depend on the capabilities of the machines: In single-stage environments with multiple machines, all machines are capable of performing the same functions, while in multi-stage systems each machine is specialized and only has specific functions, thus may only complete certain operations of a given job.

For single-stage machines, it is assumed that all machines are available for processing at the start of the schedule. Those environments are:

- **Single machine**, for $\alpha = 1$: Due to the presence of only one machine, all jobs are processed on the same machine, thus the processing time of any job $J_j,\ 1 \leq j \leq n$, is given by p_j.
- **Identical parallel machines**, for $\alpha = Pm$, $m > 1$: There are multiple machines in this environment which are identical in all aspects, thus jobs may be processed on any of the machines without their processing time being affected, hence the processing time of some job $J_j,\ 1 \leq j \leq n$, running on any machine M_i, $1 \leq i \leq m$, is given by p_j.
- **Uniform parallel machines**, for $\alpha = Qm$, $m > 1$. In this environment machines are identical in all aspects, but operate at various speeds, thus each machine M_i, is associated with a processing speed s_i. As a result the processing time of jobs is dependent on the machine on which it has been allocated, thus the processing time of some job $J_j,\ 1 \leq j \leq n,$ on machine M_i is given by p_j/s_i.
- **Unrelated parallel machines**, for $\alpha = Rm$, $m > 1$. Machines in this environment are non-identical, and perform operations at different rates. In this case, the processing time of each job depends on the machine on which it has been allocated, such that the processing time of job $J_j,\ 1 \leq j \leq n$, assigned to machine M_i is given by p_{ij}.

In the multi-stage, or shop, systems each job consist of a number of operations, and each machine is capable of processing one of these operations. This type of machine environments is representative of industrial processes such as assembly lines, where an end product has to undergo several stages of different types of processing. Variations in shop environments arise from requirements in the processing sequence of jobs and their operations. Shop environments may fall under one of the following special cases, or their extensions:

- **Open Shop,** for $\alpha = Om$: In this shop environment, each job J_j, $1 \leq j \leq n$, consists of m operations o_{ij} $(i = 1, .., m)$, where o_{ij} must be processed on machine M_i, while there are no restriction in the order in which operations are processed.
- **Flow Shop,** for $\alpha = Fm$: Similarly to open shop, jobs in the flow shop environment consist of m operations o_{ij} to be processed by machine M_i, but with restrictions regarding the sequence in which operations are processed. In this case $o_{i+1,j}$ may only begin processing on machine M_{i+1} after operation o_{ij} has completed processing on machine M_i. The precedence constraints of operations in flow shop imply that each job is required to be processed by all machines in the environment and a job may not be processed by more than one machine at a time.
- **Job Shop,** for $\alpha = Jm$: Scheduling problems in this machine environment are a generalization of flow shop problems, as in this case the number of operations for each job are arbitrary. Similarly to flow shop, there is a sequence in which the operations of each job can be performed, although the sequences may differ for different jobs.

In the *multi-stage*, or shop, systems can be further combined with any of the single stage systems, by increasing the number of machines which are capable of performing some operation. Although in this work we are only concerned with single-stage machine environments, we overview these multi-stage systems, as the solution to several single-stage problems is derived from the solutions to problems of this type.

1.1.2 Job Characteristics

In various scheduling problems, there may be various properties or limitations associated with the set of jobs. These characteristics are represented in field β of the three-field notation. For problems where multiple characteristics are associated with the set of jobs, these characteristics are represented by defining $\beta := \beta_1, \beta_2, \ldots, \beta_\xi$, where each $\beta_1, \beta_2, \ldots, \beta_\xi$ represent one of the ξ characteristics associated with the set of jobs. Field β is simply left blank if no job characteristics, other than the default ones, have been defined for the problem. Several job characteristics which are used throughout this thesis are given below:

- **Release times** (r_i): Release times indicate the point in time a job becomes available for processing, i.e. a job cannot be processed before its release time. If this value is not present in a scheduling problem's description, it is assumed that jobs are available for processing at the beginning of the schedule (jobs are released at time 0).
- **Precedence Constraints** ($prec$): When present in field β, this characteristic indicates that there is some form of dependence between jobs in the schedule. If job $J_j,\ 1 \leq j \leq n$, has precedence over job J_k, then the processing of job J_k cannot begin until job J_j is completed.
- **Preemption** ($pmtn$): Preemption indicates that the processing of a job may be interrupted at any time in order to be resumed later on the same, or different machine in the case of parallel machines. For parallel machine environments, job in preemptive schedules can be processed by only one machine at a time.
- **Splitting** ($split$): This job characteristic indicates that any single job may be processed *simultaneously* by multiple machines. Splitting are in some sense similar to preemptive jobs, differing only in the fact that jobs of this type can be processed by any number of machines at any given time.
- **Unit length jobs** ($p = 1$): This value of field β indicates that all jobs have equal processing times.

In this work, we mainly investigate scheduling problems with preemption and splitting characteristics. For preemptive problems we particularly focus on scheduling problems where the number of preemptions is limited.

1.1.3 Objective Functions

For scheduling problems the optimality criterion defined in field γ of the three field notation is given as some function of the completion time of jobs in the problem. For some job $J_j,\ 1 \leq j \leq n$, the completion time of the job is given by C_j, which represents the time at which job $J_j,\ 1 \leq j \leq n$, has satisfied its processing requirement, such that all operations of the job have completed processing. In this thesis we consider two of the most commonly encountered objective functions which are given next:

- **Makespan** ($\gamma = C_{\max}$): One of the principal objectives in machine scheduling is that of minimizing the makespan, i.e. the total duration of time in which jobs are being processed. For single machine environment the makespan objective is only of interest when there are setup times associated with the sequence in which jobs are processed; if no setup times are present the makespan of a schedule is equal to the sum of the processing times of all jobs. On parallel machines, makespan minimization can be interpreted as a fair load balancing scheme across all machines. The makespan of a schedule is given by:

$$C_{\max} = \max \{C_j \mid 1 \leq j \leq n\}.$$

- **Sum of Completion Times** ($\gamma = \sum C_j$): Another common objective criterion which has been broadly investigated in the literature. As the value of γ implies, the objective of problems of this type is the minimization of the *Total Flow Time*, given as

$$\sum_{j=1}^{n} C_j.$$

  This objective function is identical to the weighted sum of completion times ($\sum w_{ij} C_j$) objective, if there are no weights associated with assigning jobs to machines.

1.2 Complexity and Approximability

Computational problems may be considered as questions, which are defined by a set of parameters, and the relation its solution has with input variables. Initially we distinguish between a problem and a problem instance: An instance emerges from the assignment of certain values to the parameters of a problem. Solving a computational problem requires the design of an algorithm, i.e. a clear step by step procedure which produces the solution within a finite number of steps. Assessing the quality of an algorithm raises the following question: Which are the properties that we are interested in for some algorithm? The most common answer to this question is time. Indeed, one of the most desirable features of an algorithm is the time it consumes to provide a solution. The speed of an algorithm, although it may be the most important property, is certainly not the only one, as there are various other resources which may be limited in a computing machine. Similarly to time, space (memory) which an algorithm consumes can be a significant factor in the performance of some algorithm. In a broader sense, the measurement of complexity which may be of interest can vary in accordance to the requirements and nature of the problem, and the computational machine which is used. As an example, on a parallel computer, i.e. a computational machine where multiple processors are involved in the solution of the same problem, for instance by using some message passing interface, a feature of interest is the number of messages which are exchanged between processors.

Computational complexity of an algorithm is generally defined by the amount of a certain resource the algorithm consumes to solve a problem. The amount of resources consumed by an algorithm to provide a solution to some computational problem can greatly depend on the input, thus is instance related. Therefore for the majority of cases, a more reliable approach is to measure the *worst-case* complexity of an algorithm. The benefit of this approach is that worst-case complexity provides a form of guarantee of the performance of an algorithm across all instances of a problem.

Time complexity of an algorithm is given as a function $f(L)$ of the problem size L, which is an integer value representing the number of inputs (or the size of the input set). The size of a problem instance depends on the number of digits which are required to express the instance. In turn, the number of digits depends on the encoding scheme by which the problem is described. Typically, computational problems are assumed to be given in *binary encoding*, where each decimal number k is represented by $1 + \lfloor \log_2 k \rfloor$ number of digits. An alternative encoding scheme is *unary encoding*, where each number is represented by a string of 1s, thus integer k under this encoding would be represented by k digits. Regardless of the nature of the problem of interest, and the algorithm chosen for its solution, it is expected that the time required, in essence the number of steps, will increase as the problem size increases, thus larger problem instances will require more time than smaller instances. Similar functions are required for any other measurement of complexity we may be interested in. The choice of the parameter which defines the problem size, depends entirely on the nature of the problem. For example in many of the scheduling problems we document in this thesis, the problem size is equivalent to the number of jobs that require to be scheduled.

When considering the time complexity of an algorithm, we are mainly interested in the growth rate of the complexity, hence we investigate the asymptotic behavior of the algorithm for large values of L. Using the Big-O notation, time complexity is represented as $O\left(g\left(L\right)\right)$ where the relation $O\left(g\left(L\right)\right) = f\left(L\right)$ holds for some positive constant c such that $f\left(L\right) \leq cg\left(L\right)$.

Determining the complexity of a computational problem is fairly different for that of determining the complexity of an algorithm. For some computational problem, such as scheduling problems, we seek to determine the worst-case complexity of the best possible algorithm for the problem, amongst the set of all algorithms, known or unknown, which find a solution to the problem. In order to study the complexity of computational problems and algorithms, it is necessary to describe a computational model, i.e. a mathematical analogue of a computational machine which provides all the functionality resources necessary for computations we are interested in. The model adopted by complexity theory is the *Turing Machine*. This model is particularly well suited to handle a specific type of computational problems, known as *decision-making problems*; the answer to such a problem would be either "yes" or "no". It must be noted that the encoding which is used to express a problem, can have a considerable impact on its size.

The complexity classification of an algorithm and subsequently a computational problem, is determined by the limited computational resource (in essence time and space) which is required for computation using a computational model (a deterministic or non-deterministric Turing machine). For time complexity, algorithms are considered to be good if $g(L)$ is polynomial to the value of L, and is defined as a polynomial time algorithm. Computational problems for which polynomial time algorithms exist, are said to be polynomially solvable on a deterministic Turing machine. Such problems belong to complexity class P are considered to be "easy" in terms of computation.

For a large number of decision making problems, it is possible to check the validity of their solutions using a polynomial time algorithm. Problems with this characteristic are said to belong to class NP, which may also be defined as the set of all decision making problems of which the solutions can be found and verified by a *non-deterministic polynomial time* (NP) algorithm, which includes all decision making problems (problems with a yes/no answer). It is apparent that $P \subseteq NP$ as the validity of an answer for any problem in P can be also verified in polynomial time. Class NP is of particular importance as it includes a variety of useful and practical problems, for which no algorithm has been found which can provide the solution in polynomial time, and thus are considered difficult in term of computation. For such problems, a polynomial algorithm is highly unlikely to be obtained unless $P = NP$, which is the essence of the well-known P vs NP problem which was initially described in the mid 1950s, and has been of particular interest to researchers for many decades; it is generally conjectured that $P \neq NP$, although to date no proof to this statement has been given.

One of the most prominent ways of obtaining the complexity classification of a computational problem, is by using the notion of *polynomial reducibility*. This notion can be generally described as such: Let A and B be two problems with input instances I_A and I_B respectively. Then A is polynomially reducible to B if there is a algorithm R

which converts input instances of A to input instances of B in polynomial time. In other words, R has input I_A and output I_B and runs in polynomial time. The importance of a polynomial reduction lies in the following property: If there is a polynomial reduction from A to B, then A is *at most as hard* as B, thus intuitively, B is *at least as hard* as A. This notion was initially introduced by Cook (1971) and extended further by the considerable contribution of Karp (1972).

Problems in NP which can be reduced to from every other problem in that class, are termed *NP-complete*, thus may be informally stated to be at least as hard as every other problem in NP. In essence, NP-complete problems are the most difficult problems in class NP, for which no polynomial algorithm has been found. A set of problems which are known to be at least as hard as the hardest problem in NP are defined as the set of *NP-hard* problems. Note that every NP-complete problem is also NP-hard, and NP-hard problems may not necessarily be in class NP.

Polynomial solvability and NP-completeness greatly depend on the encoding scheme in which the problem is expressed. For certain problems, if the encoding is changed from binary to unary, the complexity of the problem decreases as the length of the input increases, thus reducing the restrictions on the running time of a polynomial algorithm. Optimization problems which are solved in polynomial time under unary encoding, are termed *NP-complete in the ordinary sense.* Alternatively, problems which remain NP-complete even under unary encoding, are termed *strongly NP-complete.* Similarly, an NP-hard problem is called *NP-hard in the strong sense* if its decision making version is strongly NP-complete; otherwise, the problem is called *NP-hard in the ordinary sense.*

1.2.1 Linear Programming

Linear Programming is the process of finding the optimal value of a linear function, so that the obtained value satisfies a finite set of constraints which are given in the form of linear equalities or inequalities, i.e., linear programming describes a model for maximizing or minimizing a linear function under some linear constraints. In the context of optimization, linear programming is a technique that deals with problems of allocating limited resources in a system of competing activities in an optimal way (as well as other problems of similar configuration). Considered as one of the most important mathematical discoveries of the mid 20th century, linear programming is widely used by industry.

Linear programming problems are known to be solvable in polynomial time. The proof of the complexity classification of linear programming is due to Khachiyan (1980), who showed that the ellipsoid method, initially introduced by Shor (1972), can be used for solving linear programming problems in polynomial time. Furthermore, linear programming is also extensively used for modeling and solving a large number of combinatorial problems, which may not be directly associated with the methodology. Thus, the ellipsoid algorithm, or the more recent interior points method may be used for efficiently solving many combinatorial problems. Early formulation of linear programming problems make use of the Simplex algorithm due to Dantzig et al. (1955). The simplex algorithm has proven to be very effective in solving most linear programming problems, although it has a worst-case exponential time complexity. Many of the problems, including scheduling problems we study in this thesis can be reduced to linear programming problems.

1.2.2 Integer Programming

In linear programming problems, the value of the decision making variables is allowed to be continuous, in the sense that they may take fractional values. In many, perhaps more realistic cases, fractional decision values may not be feasible. This is a characteristic of many optimization problems consisting of indissoluble entities, such as the non-preemptive problems described in Chapter 2. In such cases it may be necessary to restrict some or all of decision making variables to integer values. If all decision variables are required to be integers, the problem is called an *Integer-Programming problem* (IP). Integer programming problems are NP-hard, as a proof due to Papadimitriou (1981) demonstrates, all IP problems are reducible to the well-known NP-hard knapsack problem.

Assignment Problem:

We now provide a machine scheduling interpretation of the *Linear Assignment Problem* (LAP): Suppose that a set jobs $N = \{1, 2, \ldots, n\}$ is required to be assigned to a set of machines $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$, where $n \leq m$, in such a way that at most one job is assigned to each machine. Additionally, there is a cost c_{ij} associated with the assignment of a job J_j, $1 \leq j \leq n$, to a machine $M_i \in \mathcal{M}$. The objective is to obtain an assignment such that the total cost is minimized.

For convenience the cost values are arranged in a cost matrix $C = (c_{i,j})_{n\times m}$ such that

$$C := \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,m} \\ c_{2,1} & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ c_{n,1} & \cdots & \cdots & c_{n,m} \end{pmatrix}.$$

The objective is to obtain a set of n elements, so that exactly one element from each row and at most one element from each column are selected. This problem is also commonly referred to as the *rectangular assignment problem*, and can be expressed as the following integer programming problem:

$$\begin{array}{lll} \text{Minimize} & \sum_{j=1}^{n}\sum_{i=1}^{m} c_{ij}x_{ij} & \\ & & \\ \text{Subject to:} & & \\ & \sum_{j=1}^{n} x_{ij} \le 1 & \text{, for } \ 1 \le i \le m; \\ & \sum_{i=1}^{n} x_{ij} = 1 & \text{, for } \ 1 \le j \le n; \\ & x_{ij} \in \{0,1\} & \text{, for } \ 1 \le i \le m, \ 1 \le j \le n. \end{array} \tag{1.1}$$

Note that x_{ij} in is a binary *decision making variable* which indicates the assignment of job J_j, $1 \le j \le n$, to machine i, where $x_{ij} = 1$ if job J_j, $1 \le j \le n$, is assigned on machine M_i, and $x_{ij} = 0$ otherwise. The rectangular assignment problem is polynomially solvable. Specifically, Bourgeois and Lassale (1971) provide a polynomial time algorithm which obtains a solution to the problem as formulated in (1.1) in $O(n^2m)$ time. A special case of the problem where $n = m$, such that the problem has a square cost matrix of size $n \times n$, is solvable in $O(n^3)$ time by the Hungarian algorithm due to Kuhn (1955).

KNAPSACK PROBLEM:

Given a knapsack of a certain weight capacity W and a set of n items, where each item j has a weight w_j and a value p_j, the objective is to maximize the total value of the items which are placed in the knapsack, without exceeding its weight capacity. This problem can be expressed by the following IP formulation:

$$\begin{array}{lll} \text{Maximize} & \sum_{j=1}^{n} p_j x_j & \\ & & \\ \text{Subject to:} & & \\ & \sum_{j=1}^{n} w_j x_j \le W & \text{, for } \ 1 \le j \le n; \\ & x_j \in \{0,1\} & \text{, for } \ 1 \le j \le n; \end{array}$$

The knapsack problem is one of the combinatorial optimization problems shown to be NP-complete by Karp (1972).

Matching Problem:

Given an undirected graph G, defined by a set of vertices V, and a set of edges E, so that $G = (V, E)$, find a *matching* $\mathcal{N}$.

Definition 1.1. *A **Matching** $\mathcal{N}$, of graph G , is a subset of edges E, such that no vertex V is incident to more than one edge in $\mathcal{N}$.*

Matching problems are among the most essential problems in combinatorial optimization. A number of scheduling problems which are discussed in this thesis can be reduced to the Maximum Matching problem in Bipartite Graphs.

Definition 1.2. *A matching $\mathcal{N}$ is said to be **Maximum** if for any other matching $\mathcal{N}'$, $|\mathcal{N}| \geq |\mathcal{N}'|$.*

Definition 1.3. *A graph G in which the set of vertices can be divided into two disjoint subsets, is termed a **Bipartite Graph.***

Maximum Bipartite Matching:

Given a bipartite graph consisting of sets A and B, and set of edged E, so that $G = (A \cup B, E)$, find a maximum matching $S \subset A \times B$.

A matching gives an assignment of nodes in A to nodes in B. Each element of A may be matched to certain elements in B, with the objective of maximizing the number of matchings. This problem has a polynomial reduction to the well-known *Network Flow* problem, which is solvable in $O(mn)$ time by the algorithm due to Ford and Fulkerson (1955).

1.2.3 Approximation

There are various approaches to dealing with computational problems which are considered hard, i.e. problems for which it is highly unlikely that a polynomial time algorithm which provides the solution can be found and are often infeasible to solve to optimality within a reasonable amount of time. An approach to obtaining a solution for such problems comes in the form of heuristic algorithms, which typically provide fast, practical solutions to some computationally hard problems, often in linear time, and satisfy certain practical requirements. The downside of heuristics is that solution quality is often sacrificed for the speed in which they can be provided, and as such, answers obtained by heuristic algorithms might greatly deviate from an optimal solution. For computationally hard problems it is often desirable to have a provable guarantee of the quality of the answer. This has lead to extensive research in *approximation algorithms* which obtain an approximate solution within a factor of the optimal, and do so in polynomial time.

Consider a scheduling problem to minimize a cost function $\Phi(S) \geq 0$, where S denotes any feasible schedule for that problem. For a schedule S^H obtained by an approximation algorithm H and an optimal schedule S^*, the *approximation ratio* of algorithm H is given by $\Phi\left(S^H\right)/\Phi\left(S^*\right)$. We denote by ρ the *worst case ratio*, or *ratio guarantee* of algorithm H such that

$$\frac{\Phi\left(S^H\right)}{\Phi\left(S^*\right)} \leq \rho.$$

As the approximation ratio of algorithm H is bounded by ρ, where $\rho \geq 1$, we know that any solution obtained by H is guaranteed to be *at most* ρ times the optimal value. Thus, H is called an *ρ-approximation algorithm*.

Definition 1.4. *If there is an approximation algorithm for some NP-complete problem such that ρ remains constant across all possible instances, the algorithm is referred to as a* ***constant-factor approximation algorithm.***

An approximation scheme for an optimization problem is defined by an approximation algorithm H_ϵ, which receives as input a problem instance and a value $\epsilon > 0$, and returns an ϵ-approximate solution in polynomial time with respect to ϵ. A *Polynomial Time Approximation Scheme* (PTAS) is a set of algorithms, which given any problem instance and any positive ϵ, provide an ϵ-approximate solution in polynomial time, with respect to the size of the problem instance, thus the approximation ratio can be as small as $1+\epsilon$. If the solution from a PTAS is obtained in time which is also polynomial with respect to $1/\epsilon$, then it is defined as a *Fully Polynomial Time Approximation Scheme* (FPTAS).

For NP-hard problems, obtaining a PTAS or an FPTAS is the ideal result of approximation, but is not always possible. In order to determine if a PTAS exits for a given NP-hard problem, Papadimitriou and Yannakakis (1988) introduce the concept of *linear reduction* (L-reduction). An L-reduction of optimization problems A to a B has the following characteristics:

- There is a function R which transforms instances of A into instances of B, such that $R\left(I_A\right) = I_B$, where I_A and I_B are input instances of A and B respectively, such that the inequality

$$\Phi\left(S^*\left(I_B\right)\right) \leq a \cdot \Phi\left(S^*\left(I_A\right)\right),$$

holds for some positive constant a, where $\Phi\left(S^*\left(I\right)\right)$ is the optimum cost (cost of the optimum solution) of instance I.

- For any feasible solution $S\left(I_B\right)$, with cost $\Phi\left(S\left(I_B\right)\right)$, there is a feasible solution $S\left(I_A\right)$, such that

  $$\left|\Phi\left(S\left(I_A\right)\right)-\Phi\left(S^*\left(I_A\right)\right)\right| \leq b\cdot\left|\Phi\left(S\left(I_B\right)\right)-\Phi\left(S^*\left(I_B\right)\right)\right|,$$

  for some positive constant b.

Papadimitriou and Yannakakis (1988) prove that if an optimization problem A is reducible to an optimization problem B for which there is a polynomial approximation algorithm with performance guarantee $\rho = 1+\epsilon$, then there is a polynomial time approximation algorithm for A with $\rho = 1+ab\epsilon$. Which implies that if there is a PTAS for B then there is also a PTAS for A.

The complexity class of approximable (APX) problems, includes all problems for which there is a constant factor approximation algorithm. The most difficult problems in this class are the *APX-complete* problems, for which no PTAS has been found, and is unlikely to be found unless $P = NP$. Similarly to the earlier description of class hardness, problems which are at least as hard as an APX-complete problem, are called APX-hard.

Subset-sum Problem:

Consider the following decision version of the Subset-sum problem, as described by Garey and Johnson (1979): Given a positive integer E, a set $R = \{1, 2.\ldots, r\}$, and a positive integer e_j, for each $j \in R$, does there exist a subset $R' \subseteq R$ such that $\sum_{j\in R'} e_j = E$?

The optimization version of the Subset-sum problem can be defined as

$$\begin{array}{lll} \text{Maximize} & \sum_{j=1}^{r} e_j x_j & \\ & & \\ \text{Subject to:} & & \\ & \sum_{j=1}^{r} e_j x_j \leq E & \text{, for } 1 \leq j \leq r; \\ & x_j \in \{0,1\} & \text{, for } 1 \leq j \leq r, \end{array} \tag{1.2}$$

where E and e_j, $j \in R$, are all positive integers. Since the decision version of the above problem is an NP-complete problem, the Subset-sum problem of the form (1.2) is known to be NP-hard in the ordinary sense. This problem allows an FPTAS which, for a given positive ε, either finds an optimal solution $x_j^* \in \{0,1\}$, $j \in N$, such that

$$\sum_{j\in R} e_j x_j^* < (1-\varepsilon)E,$$

or finds an approximate solution $x_j^\varepsilon \in \{0,1\}$, $j \in R$, such that

$$(1-\varepsilon)E \leq \sum_{j\in R} e_j x_j^\varepsilon \leq E.$$

The fastest known FPTAS is due to Keller et al. (2003), which requires no more than $O\left(\min\left\{n/\varepsilon, n + \frac{1}{\varepsilon^2}\log\left(\frac{1}{\varepsilon}\right)\right\}\right)$ time.

CHAPTER 2

Scheduling on Parallel Machines: Complexity and Approximation

For the problems for minimizing the makespan and the total flow time on identical, uniform and unrelated machines, in this chapter we provide a review of the known results of exact and approximate solutions to these problems. Throughout each of the following sections, without loss of generality, is it assumed that jobs are numbered according to the *longest processing time first* (LPT) rule, where jobs are numbered in non-increasing order of their processing times so that

$$p_1 \geq p_2 \geq \ldots \geq p_n. \tag{2.1}$$

Alternatively, certain problems require jobs to be ordered according to the *shortest processing time first* (SPT) rule. If jobs are numbered according to the SPT rule, then they are numbered in non-decreasing order of their processing times, so that

$$p_1 \leq p_2 \leq \ldots \leq p_n \tag{2.2}$$

For a set of jobs $Q \subseteq N$, define

$$p(Q) = \sum_{J_j \in Q} p_j.$$

In particular, $p(\varnothing) = 0$ and

$$p(N) = \sum_{j=1}^{n} p_j.$$

2.1 Identical Parallel Machines

2.1.1 Minimizing Makespan

Consider the problem of obtaining a schedule S^* with the minimum makespan on m identical parallel machines. A schedule S for this problem is defined by a partition of the set of jobs N into m subsets $N_1, N_2, \ldots, N_m$, so that jobs in N_i are scheduled on machine M_i. Let $p(N_i)$ represent the sum of processing times of jobs in N_i. In this case, the value of makespan for a non-preemptive schedule S_{np} is given by

$$C_{\max}(S_{np}) = \max\{p(N_i) \mid 1 \le i \le m\}.$$

It is easy to verify that for any feasible schedule S, the makespan is at least as long as the processing time of any job, so that

$$C_{\max}(S) \ge p_j,\ 1 \le j \le n. \tag{2.3}$$

Due to the assumption that jobs are always numbered according to the LPT rule in (2.1), bound (2.3) is simply given as

$$C_{\max}(S) \ge p_1. \tag{2.4}$$

Consider a schedule where all machines are busy processing some job throughout the duration of the schedule, such that no machine becomes idle before all jobs are processed. In this case, the total processing time $p(N_i)$ of every machine M_i is given by the *average machine load* T, where

$$T = \frac{1}{m} p(N). \tag{2.5}$$

It is easy to see that (2.5) is a lower bound for the makespan of any feasible schedule, so that

$$C_{\max}(S^*) \ge T. \tag{2.6}$$

Thus, due to (2.4) and (2.6), the general expression for the lower bound for the value of the makespan is given by

$$C_{\max}(S^*) \ge \max\{p_1, T\}. \tag{2.7}$$

This bound holds for both the preemptive and non-preemptive versions of this problem, while for the preemptive version, i.e. problem $Pm\,|pmtn|\,C_{\max}$, this bound is tight for any optimal schedule.

An optimal schedule for problem $Pm\,|pmtn|\,C_{\max}$ is obtained in polynomial time by applying the algorithm due to McNaughton (1959) to any instance of this problem. This algorithm, which is commonly referred to as *McNaughton's wrap-around method*, considers the segmentation of the total processing time $p(N)$ into m intervals. The length of these intervals depends on where the maximum is achieved on the right hand side of (2.7). A description of the algorithm by McNaughton is given next.

Algorithm WrapAround [McNaughton (1959)]

INPUT: an instance of problem $Pm\,|pmtn|\,C_{\max}$

OUTPUT: An optimal preemptive schedule S_p^*

Step 1. Calculate bound $c = C_{\max}(S^*)$ in accordance to (2.7).

Step 2. Form a temporary single-machine schedule S_1, by assigning jobs in an arbitrary sequence to a hypothetical machine $A \notin \mathcal{M}$, in the time interval $I := [0, p(N)]$.

Step 3. Split S_1 into m pieces, such that the first $m-1$ pieces have length c and the m-th piece has length $p(N) - (m-1)c$. Assign the jobs, and job fractions processed on A in the time interval $I_i = [(i-1)c, (i \cdot c)]$ to be processed on machine M_i for $1 \le i \le m-1$. Assign jobs and job fractions which are processed on A in the time interval $I_m = [(m-1)c, p(N)]$, to be processed on machine M_m.

Step 4. Stop.

Algorithm WrapAround finds an optimal preemptive schedule in $O(n+m)$ time, with at most $m-1$ preempted jobs.

For further purposes, it is convenient to introduce the following classification of the instances of the problem.

Definition 2.1. *An instance of the problem on m parallel identical machines is said to belong to Class 1, if $C_{\max}\left(S_p^*\right) = p_1$; otherwise, it said to belong to Class 2.*

For the non-preemptive version of this problem, obtaining a feasible schedule is NP-hard. Specifically, for the problem with two identical machines, Lenstra and Kan (1979) demonstrate that problem $P2\,|\,|\,C_{\max}$ can be polynomially reduced from problem PARTITION described in Section 1.2. It follows that for $m \ge 3$, this problem is strongly NP-hard, as it can be polynomially reduced from the 3-PARTITION problem. As a result, it is unlikely that this problem will be solved in polynomial time. Because problem $Pm \,|\,|\, C_{\max}$ has no known algorithm which can find a feasible schedule with optimal makespan in polynomial time, approximation algorithms for this problem have been extensively studied.

The earliest approximation is the *List Scheduling* (LS) algorithm due to Graham (1966). The algorithm follows the process of initially creating a list $\mathcal{L}$ of jobs, then sequentially assigning each job to the first available machine, whilst removing each assigned job from the list. The algorithm terminates when there are no more jobs left in $\mathcal{L}$. Graham's List Scheduling algorithm is described next.

Algorithm LS [Graham (1966)]

INPUT: an instance of problem $Pm \,|\,|\, C_{\max}$

OUTPUT: A non-preemptive schedule S^{LS}

Step 1. Form an arbitrary ordered list $\mathcal{L}$ of jobs.

Step 2. When a machine becomes available, take the first job in $\mathcal{L}$ and assign it to the machine. Remove the assigned job $\mathcal{L}$.

Step 3. While there are jobs left in $\mathcal{L}$, repeat Step 2.

Step 4. Stop.

Algorithm LS finds a feasible schedule S^{LS} in $O(nm)$ time. Furthermore, the makespan of schedule S^{LS} is shown to be at most $2-1/m$ times the optimal makespan. The proof for the worst-case performance ratio of Algorithm LS is shown in the following theorem due to Graham (1969).

Theorem 2.1 (Graham (1969))**.** *For problem $Pm \,|\,|\, C_{\max}$, Algorithm LS finds a schedule S^{LS} such that the bound*

$$\frac{C_{\max}\left(S^{LS}\right)}{C_{\max}\left(S^{*}\right)} \leq 2 - \frac{1}{m}, \tag{2.8}$$

holds, and this bound is tight.

Proof: Assume that J_k is the job that terminates schedule S^{LS}. Job J_k starts processing at time t and completes at time $C_k = C_{\max}$. No machine is idle before time

$t = C_{\max}\left(S^{LP}\right) - p_k$, hence we deduce that $p(N) \geq mt + p_k$. Thus from (2.6) we obtain a lower bound

$$C_{\max}(S^*) \geq t + \frac{p_k}{m}. \tag{2.9}$$

Due to (2.3) and (2.9), we have that

$$\begin{aligned} C_{\max}(S^{LS}) &= t + p_k = \left(t + \frac{p_k}{m}\right) + \left(\frac{m-1}{m}\right) p_k \\ &\leq C_{\max}(S^*) + \frac{m-1}{m} C_{\max}(S^*), \end{aligned}$$

as required.

The tightness of (2.8) can be shown by considering an instance of $Pm\,|\,|\,C_{\max}$ where there are $n = m(m-1) + 1$ jobs, such that $p_1 = p_2 = \ldots = p_{n-1} = 1$ and $p_n = m$. Constructing the list in the order of its numbering, algorithm LS creates a schedule S^{LS} such that $C_{\max}\left(S^{LS}\right) = 2m - 1$, while an optimal assignment of jobs to machines exists which yields a schedule S^* where $C_{\max}(S^*) = m$. Therefore

$$\frac{C_{\max}\left(S^{LS}\right)}{C_{\max}(S^*)} = 2 - \frac{1}{m}.$$

for illustration, consider a tight instance of this problem for $m = 3$. The structure of an optimal schedule S^* for this instance is shown in the following table.

p_1	p_2	p_3	p_4	p_5	p_6	p_7	$C_{\max}(S^*)$	S^*		
								N_1	N_2	N_3
1	1	1	1	1	1	3	3	7	$1,2,3$	$4,5,6$

(2.10)

An application of Algorithm LS to this instance yields a schedule S^{LS} of the following structure.

p_1	p_2	p_3	p_4	p_5	p_6	p_7	$C_{\max}(S^{LS})$	S^{LS}		
								N_1	N_2	N_3
1	1	1	1	1	1	3	5	$1,4,7$	$2,5$	$3,6$

(2.11)

□

A version of the list scheduling algorithm due to Graham (1969), where the elements of the list $\mathcal{L}$ are sorted according to the LPT rule in (2.1). This algorithm is shown next:

Algorithm LPT-LS [Graham (1966)]

INPUT: an instance of problem $Pm \,|\,|\, C_{\max}$

OUTPUT: A non-preemptive schedule S^{LPT}

Step 1. Form a list of jobs $\mathcal{L}$, sorted in non-increasing order of their processing speeds, according to (2.1).

Step 2. When a machine becomes available, take the first job in $\mathcal{L}$ and assign it to the machine. Remove the assigned job $\mathcal{L}$.

Step 3. While there are jobs left in $\mathcal{L}$, repeat Step 2.

Step 4. Stop.

Note that the only difference between LPT-LS and LS algorithms is found in Step 1. Algorithm LPT-LS is shown to deliver an improved worst-case performance ratio in comparison to an arbitrarily ordered list. This is demonstrated next:

Lemma 2.1. *Suppose that in a schedule S^{LPT}, job J_j, $1 \leq j \leq n$, is sequenced as the $h-th$ job on the machine of its allocation. Then*

$$p_j \leq \frac{1}{h} C_{\max}\left(S^*\right). \tag{2.12}$$

Theorem 2.2. *For some job $J_k \in N$ which terminates schedule S^{LPT}, and is sequenced as the $h-th$ job on the machine of its allocation, then*

$$\frac{C_{\max}\left(S^{LPT}\right)}{C_{\max}\left(S^*\right)} \leq \frac{h+1}{h} - \frac{1}{hm}, \tag{2.13}$$

holds, and this bound is tight for $h \geq 3$.

Proof: Suppose that job J_k begins processing at time t. Due to Lemma (2.1),

$$p_k \leq \frac{1}{h} C_{\max}\left(S^*\right).$$

The bound (2.9) holds, which gives

$$\begin{aligned} C_{\max}\left(S^{LPT}\right) &= t + p_k = \left(t + \frac{p_k}{m}\right) + \frac{m-1}{m} p_k \\ &\leq C_{\max}\left(S^*\right) + \left(1 - \frac{1}{m}\right) \frac{1}{h} C_{\max}\left(S^*\right), \end{aligned}$$

which proves Lemma (2.1). □

This bound is tight for all $h \geq 3$. For $h = 2$, the bound is given by

$$\frac{C_{\max}\left(S^{LPT}\right)}{C_{\max}\left(S^{*}\right)} \leq \frac{4}{3} - \frac{1}{3\left(m-1\right)}.$$

Thus applying (2.13) with $h = 3$, yields the inequality

$$\frac{C_{\max}\left(S^{LPT}\right)}{C_{\max}\left(S^{*}\right)} \leq \frac{4}{3} - \frac{1}{3m},$$

which gives a global worst-case bound on the performance of LPT.

The tightness of this bound is demonstrated by considering an instance for $Pm \mid \mid C_{\max}$:

$$\begin{aligned}
p_1 &= p_2 &&= 2m-1\\
p_3 &= p_4 &&= 2m-2\\
&\dots\\
p_{2k-1} &= p_{2k} &&= 2m-k\\
&\dots\\
p_{2m-1} &= p_{2m} &&= m\\
p_{2m+1} &= m.
\end{aligned}$$

Using this problem instance, an LPT schedule S^{LPT} can be obtained, such that $C_{\max}\left(S^{LPT}\right) = 4m-1$, while the optimal schedule S^{*}, due to (2.7), yield makespan $C_{\max}\left(S^{*}\right) = 3m$. Therefore we obtain

$$\frac{C_{\max}\left(S^{LPT}\right)}{C_{\max}\left(S^{*}\right)} = \frac{4}{3} - \frac{1}{3m}.$$

For $m = 3$, algorithm LPT-LS yields a schedule S^{LPT} □

A variant of Algorithm LS suggested by Graham (1969), schedules the k longest jobs optimally, then applies Algorithm LS to the remaining jobs. This approach gives a ratio guarantee of $1 + (1 - 1/m) \,/\, (1 + \lfloor k/m \rfloor)$. Thus for a fixed number of machines, there is a PTAS for different values of k, although this yields $O\left(n^{km}\right)$ running time.

Another approach to approximating a solution for $Pm \mid \mid C_{\max}$, is by considering it as a *Bin-Packing* problem. The bin-packing problem can be described as such: Given a set of bins $\mathcal{U} := \{U_1, U_2, \dots, U_m\}$ where each bin U_i has capacity u_i, and a list of elements $\mathcal{L} := \{1, 2, \dots, n\}$ where each element j has size v_j. The objective of this problem is to find an arrangement of items into bins which uses the minimum number of bins. It may be apparent at this stage, that bin packing and the problem of minimizing the makespan share similar objectives.

The Multifit heuristic due to Coffman (1993), finds via binary search the minimum capacity of the m bins into which n items can be placed using the *first-fit decreasing* (FFD) heuristic, where in each iteration of the algorithm, the largest unassigned item into the first bin in which it fits. For a number of k binary search iterations, Multifit is shown to have a ratio guarantee of $\rho + 2^{-k}$, where $\rho \leq 1.22$. The value of ρ for Multifit is improved to 1.2 due to Friesen (1987).

2.1.2 Minimizing Sum of Completion Times

Consider the problem of finding a schedule which minimizes the sum of completion times, denoted by $P \,|\,|\, \sum C_j$. A solution to this problem is defined by a partition of N into m disjoint subsets $N_1, N_2, \dots, N_m$, where N_i is the set of jobs to be processed by machine M_i. Furthermore, every machine M_i is associated with a list $\mathcal{L}_i$, which determines the sequence in which jobs are processed by M_i.

Let t_j denote the contribution of the processing time of job J_j, $1 \leq j \leq n$, to the value of the objective function. The contribution of a job for Identical machines is determined by the position from last in which the job is processed by a machine. Consider the following example: A machine M_k, processed its set of jobs N_k in the sequence given by $\mathcal{L}_k := \{J_1, J_2, J_3\}$, where the processing time of J_r, $r \in N_k$, is given by p_r. As J_1 is processed in the third position from the rear of $\mathcal{L}_k$, we have that $t_1 = 3$. The contribution of the rest of the jobs is similarly determined.

The contribution of a job defines the number of times its processing requirements will be present in the total value of the objective function, so that the value of the objective function is given by

$$\sum_{j=1}^{n} C_j = \sum_{j=1}^{n} t_j p_j.$$

In the single machine version of this problem denoted by $1 \,|\,|\, \sum C_j$, the value of the objective function is minimized if jobs are scheduled in the SPT (2.2) order. Hence, if all jobs in N are added to list $\mathcal{L}$ in the SPT order, then the first job will have the shortest processing time, so that the order of $\mathcal{L}$ is given by

$$\mathcal{L} := \left\{J_n, J_{(n-1)}, \dots, J_1\right\}, \tag{2.14}$$

where jobs are numbered according to LPT (2.1). In this case, for job J_j, $1 \leq j \leq n_j$, has contribution $t_j = (n - j) + 1$.

For identical parallel machines, Conway et al. (1967) present a modified version of Algorithm LS, where the list of jobs in Step 1 of the algorithm is sorted according to the SPT rule. This algorithm constructs an optimal schedule S^* where each job is assigned to machine $(j-1) \bmod (m)$, and the contribution of each job J_j, $1 \le j \le n_j$ is given by $t_j = \lceil j/m \rceil$. Thus the value of the objective function is given by

$$\sum_{j=1}^{n} C_j\left(S^*\right) = \sum_{j=1}^{n} p_j \left\lceil \frac{j}{m} \right\rceil.$$

The running time of the modified Algorithm LS by Conway et al. (1967) is $O(n \log n)$, where $O(n \log n)$ time is required for sorting jobs in the SPT order.

Allowing preemption for this problem does not reduce the value of the objective function. The proof of this statement is due to McNaughton (1959), who shows that for the problem of minimizing the weighted sum of completion times, denoted by $P||\sum w_j C_j$ introducing preemption does not yield a schedule which improves the value of the objective function. Hence for identical machines, interest is only drawn towards the non-preemptive version of this problem.

2.2 Uniform Parallel Machines

2.2.1 Minimizing Makespan

Consider the problem on uniform machines. An instance I of this problem is defined by a list of n processing times $\mathcal{P}_n$, and a set of m machines $\mathcal{M}_m$, so that $I = (\mathcal{P}_n, \mathcal{M}_m)$. As discussed in Section 1.1, in the uniform machine environment, each machine M_i is associated with a processing speed s_i. Without loss of generality, in this section and throughout the rest of this thesis we assume that machines are numbered according to the *fastest machine first* (FM) rule, i.e., machines are numbered in non-increasing order of their processing speeds, so that

$$s_1 \ge s_2 \ge \ldots \ge s_m. \tag{2.15}$$

Given an instance $I = (\mathcal{P}_n, \mathcal{M}_m)$, for each u, $1 \le u \le m$, we denote the total speed of the u fastest machines by $s(\mathcal{M}_u)$, where

$$s(\mathcal{M}_u) = \sum_{i=1}^{u} s_i.$$

In the preemptive version of this problem, i.e., problem $Qm\left|pmtn\right|C_{\max}$, for some instance I where $n \geq m$, as shown in Brucker (2007), a necessary condition for processing all jobs in the interval $[0, T]$, is

$$p\left(N\right) = p_1 + p_2 + \ldots + p_n \leq s_1 T + s_2 T + \ldots + s_m T = s\left(\mathcal{M}\right) T$$

or

$$\frac{p\left(N\right)}{s\left(\mathcal{M}\right)} \leq T.$$

Furthermore, as $p\left(N_i\right)/s\left(\mathcal{M}_i\right)$ is a lower bound on the length of a schedule for the jobs $J_1, \ldots J_i$, the inequality

$$\frac{p\left(N_i\right)}{s\left(\mathcal{M}_i\right)} \leq T,\ 1 \leq j \leq (m-1)$$

must hold. Thus, the makespan of an optimal preemptive schedule $S_p^*\left(I\right)$ is given by

$$C_{\max}\left(S_p^*\left(I\right)\right) = \max\left\{T_u | 1 \leq u \leq m\right\}, \tag{2.16}$$

where T_u is the average machine load obtained by assigning the u largest jobs on the u fastest machines, so that

$$T_u = \frac{p\left(N_u\right)}{s\left(\mathcal{M}_u\right)}, \tag{2.17}$$

where

$$T_m = \frac{p\left(N\right)}{s\left(\mathcal{M}\right)}. \tag{2.18}$$

Note that for the case $n < m$, only the n fastest machines need to be considered.

For further purposes, it is convenient to introduce the following classification of the instances of the problem.

Definition 2.2. *An instance of the problem on m parallel uniform machines is said to belong to Class u, if $C_{\max}\left(S_p^*\right) = T_u$, where $1 \leq u \leq m$.*

An optimal schedule for problem $Qm\left|pmtn\right|C_{\max}$, can be found in polynomial time. A plethora of algorithms are present throughout the literature which solve this problem in various degrees of speed. An optimal schedule is obtained via an adaptation of McNaughton's wrap-around algorithm. Following the generalization of the lower bounds described by McNaughton for the problem on identical machined, to the case of uniform machines, Hovarth, Lam and Sethi (1977) provide the *longest remaining processing time on fastest machine* (LRPT-FM) rule, where at any given point the job with the longest remaining processing time is assigned to the fastest available machine.

The level $p_j(t)$ of a job $J_j,\ 1 \leq j \leq n$, is defined as the remaining processing time of the job after time t. At time t, procedure Assign(t) is called, which obtains a partial assignment of jobs to machines. This partial assignment remains until a time $f > t$ is reached, at which point the process is repeated.

Procedure Assign(t) [Hovarth, Lam and Sethi (1977)]

INPUT: A set of jobs $J \subseteq N$, where $p_j(t) > 0$, for every $j \in J$, and a set of machines $\mathcal{M}$

OUTPUT: A partial assignment of jobs to machines up to time t.

Step 1. Find set L, so that $L := \{j \mid p_j(t) = \max\{p_j(t) \mid 1 \leq j \leq n\}\}$

Step 2. Define $r := \min\{|\mathcal{M}|, |L|\}$

Step 3. Assign jobs in L to be jointly processed on the r fastest machines in $\mathcal{M}$.

Step 4 Define $J := J \setminus L$

Step 5. Remove the r fastest machines from $\mathcal{M}$.

Algorithm LRPT-FM [Hovarth, Lam and Sethi (1977)]

INPUT: an instance I of problem $Qm\,|pmtn|\,C_{\max}$.

OUTPUT: An optimal preemptive schedule $S_p^*(I)$.

Step 1. Define $t = 0$;

Step 2. Run procedure Assign(t);

Step 3. Determine $t_1 = \min\{f \mid f > t\}$ so that there is a job which terminates at time s.

Step 4. Determine $t_2 = \min\{f \mid f > t\}$ so that there are jobs k, l with $p_k(t) > p_l(t)$ and $p_k(f) > p_l(f)$

Step 5. Define $t = \min\{t_1, t_2\}$;

Step 6. Repeat Step 2 while there is a job $J_j,\ 1 \leq j \leq n$, so that $p_j(t) > 0$

Step 7. Construct the schedule

Step 8. Stop.

As an example, consider a case with two machines M_1 and M_2, with corresponding speeds s_1 and s_2 such that $s_1 > s_2$. At the time which M_1 completes the processing of a job at time t, any job which has not completed processing by that time on machine M_2, is transferred to M_1. In a more general case with m machines, every time some machine M_l completes processing a job at time t, the longest job which is being processed by any of the slower machines, is transferred to M_l.

The LRPT-FM based algorithm obtains an optimal solution in $O\left(mn^2\right)$ time, and requires a virtually unlimited number of preemptions. A more case-based algorithm is proposed by Gonzalez and Sahni (1978), which not only obtains an optimal solution in a considerably improved $O(n + m \ \log m)$ time, but also requires a maximum of $2\,(m-1)$ preemptions in order to construct a schedule.

As a generalization of the problem on identical machines, problem $Qm \,|\,| C_{\max}$ is NP-hard. As a result, various approximation techniques are applied to provide considerably satisfactory solutions. Of those, an adaptation of the LPT list scheduling algorithm has been one of the earliest approaches to approximating the solution. The modifications to this algorithm are designed to accommodate the variance of speeds in uniform machine environments. In this algorithm due to Liu and Liu (1974), the list of jobs $\mathcal{L}$ is initially ordered according to the LPT rule (2.1). Jobs are assigned to machines by assigning the first unassigned job in $\mathcal{L}$ to the fastest available machine.

Algorithm QLPT [Liu and Liu (1974)]

INPUT: An instance of problem $Qm \,|\,| C_{\max}$.

OUTPUT: A non-preemptive schedule S_{LPT}.

Step 1. Form a list of jobs $\mathcal{L}$ in accordance to the LPT rule.

Step 2. Assign the first job in $\mathcal{L}$ to the machine on which it has the earliest completion time. Remove that job from $\mathcal{L}$

Step 3. Repeat Step 2 until all jobs in $\mathcal{L}$ have been scheduled.

Algorithm QLPT requires $O\left(n \ \log n + nm\right)$ time, where $O\left(n \ \log n\right)$ time is required for sorting the list.

The performance of LPT on uniform machines has been a subject of academic interest for several decades, with incremental improvements of the bounds due to Gonzalez, Ibarra and Sahni (1977),Dobson (1984) and Friesen (1987). The best known results on the worst-case ratio are due to Kovács (2006), who shows that

$$1.54 \leq \rho_{LPT} \leq 1.577.$$

2.2.2 Minimizing Sum of Completion Times

The problem of minimizing the sum of completion times on uniform machines, denoted by $Qm\,||\sum C_j$, closely resembles its counterpart on identical machines, and is solvable in polynomial time. The solution we explore in this section is due to Brucker (2007). Similarly to the identical machine case, the algorithm which is used for finding an optimal schedule, takes into account the contribution of the processing time of each job in the objective function. This is achieved by considering a schedule as a partition of the set of jobs N in m disjoint sets $N_1, N_2, \ldots, N_m$ which define the set of jobs to be processed by each of the m machines. Furthermore, there is a sequence I_i of jobs associated with each N_i. In this case, for a sequence of r jobs I_i such that $I_i = (J_{j_1}, J_{j_2}, \ldots, J_{j_r})$, the contribution to the objective function from the jobs assigned to machine M_i is given by

$$p_{j_1}\frac{r}{s_i} + p_{j_2}\frac{r-1}{s_i} + \ldots + p_{j_r}\frac{1}{s_i},$$

which indicates that in an optimal schedule, the jobs of set N_i must be scheduled on M_i according to the SPT rule. The contribution of the processing time of a job $J_j,\ 1 \le j \le n$, in this problem is denoted by t_j, such that

$$t_j = \frac{k}{s_i},$$

where k indicates the position from the end of a sequence in which a job is assigned to a machine. In order to construct an optimal schedule, it is only necessary to obtain a non-decreasing sequence of the n smallest of t_j from the set

$$\left\{\frac{1}{s_1}, \frac{1}{s_2}, \ldots, \frac{1}{s_m}, \frac{2}{s_1}, \frac{2}{s_2}, \ldots, \frac{2}{s_m}, \frac{3}{s_1}, \ldots\right\},$$

Such that, if $t_j = \frac{k}{s_i}$, then job $J_j,\ 1 \le j \le n$, is scheduled as the k-th last job on machine M_i due to jobs being numbered in non-increasing order of their processing speeds, i.e. $p_1 \ge p_2 \ge \ldots \ge p_n$. The following algorithm obtains an optimal solution for the non-preemptive problem of minimizing the makespan in $O(n \log \max\{n, m\})$ time.

Algorithm Q-NonPreempt (Brucker (2007))

INPUT: An instance of problem $Q\,|\,|\sum C_j$.

OUTPUT:An partition of set N in m sequenced sets $N_1, N_2, \ldots, N_m$ which define and optimal non-preemptive schedule S^*_{np}.

Step 1. Scanning machines in the order of their numbering, for each machine i define an empty sequence I_i, and define $w_i = \frac{1}{s_i}$.

Step 2. Find the largest machine index z with $w_z = \min\{w_i | 1 \leq i \leq m\}$, and append job j to the last available position in sequence I_i. Define $w_i = w_i + \frac{1}{s_j}$.

Step 3. Repeat Step 2 until all jobs have been scheduled.

The solution to the preemptive problem of minimizing the sum of completion times on uniform machines, denoted by $Qm\,|pmtn|\sum C_j$, is obtained via an algorithm which utilizes an adapted version of the SPT rule. In this case, jobs are initially sorted in non-increasing order of their processing times and each successive job is scheduled preemptively such as to minimize its completion time. Thus, job n is initially scheduled on the fastest machine M_1 until its completion time $C_1 = p_n/s_1$, then job $n-1$ is processed by machine M_2 in the time interval $[0, C_1]$, until its preempted and the remaining processing time being assigned to machine M_1, etc. In essence, this algorithm begins by assigning the smallest job on the fastest machine in the schedule, then incrementally shifting the remaining processing time of jobs towards the fastest machines.

Algorithm Q-SPT [Labetoulle et al. (1982)]

INPUT: An instance of problem $Q\,|pmtn|\sum C_j$.

OUTPUT: An optimal preemptive schedule S_p^*.

Step 1. Define time $\alpha = 0$, and determine the smallest job k which has not fulfilled its processing requirement p_k.

Step 2. Define time $t = p_k/s_1$.

Step 3. Schedule every job $\nu \leq k$ and $\nu \geq \max\{1, i-m+1\}$, on machine $M_{1+k-\nu}$ for the time interval $[\alpha, \alpha+t]$, and reduce the remaining processing requirement of job ν by $t \cdot s_{1+k-\nu}$, such that $p_\nu = p_\nu - t \cdot s_{1+k-\nu}$.

Step 4. Define $\alpha := \alpha + t$. Repeat Step 2 until job 1 has fulfilled its processing requirement.

Algorithm Q-SPT due to Labetoulle et al. (1982), finds an optimal schedule in $O(n \log n + mn)$ time, requiring at most $(m-1)(n-m/2)$ preemptions.

2.3 Unrelated Parallel Machines

2.3.1 Minimum Makespan In Open Shop

Consider the preemptive open shop problem of minimizing the makespan, denoted by $O\left|pmtn\right|C_{\max}$, as described in Section 1.1. This problem plays an auxiliary role in our consideration of the problem on unrelated parallel machines. Recall that in open shop problems, every job consists of a number of operations. Furthermore, each operation o_{ij} of a job $J_j,\ 1 \leq j \leq n$, may be only processed by a machine M_i. The processing time of operation o_{ij} is denoted by p_{ij}. It is easy to see that the amount of time a job $J_j,\ 1 \leq j \leq n$, will be present in the system, is determined by the sum of processing times of its operation. When preemptions are allowed, the lower bound for the value of the makespan is obtained by observing the conditions for feasibility of a schedule. In this case, a feasible schedule S may not terminate before any job in the schedule has fulfilled its processing requirement on all machines, thus for a feasible schedule S, the relation

$$C_{\max}\left(S\right) \geq \max\left\{\sum_{i=1}^{m} p_{ij}|j \in n\right\},$$

holds. Additionally, in any schedule S, jobs cannot be completed on all machines earlier than any machine can complete all of its jobs, thus

$$C_{\max}\left(S\right) \geq \max\left\{\sum_{j=1}^{n} p_{ij}|1 \leq i \leq m\right\}.$$

Due to the above, the general lower bound of the makespan for the preemptive open shop problem is given by

$$C_{\max}\left(S\right) \geq \max\left\{\max\left\{\sum_{i=1}^{m} p_{ij}|j \in n\right\}, \max\left\{\sum_{j=1}^{n} p_{ij}|1 \leq i \leq m\right\}\right\}, \tag{2.19}$$

and this bound is tight.

Throughout the literature, various algorithms have been used to solve $Om\left|pmtn\right|C_{\max}$. Gonzalez and Sahni (1976) provide two algorithms for this problem, the first of which utilizes maximum edge matchings in bipartite graphs, and obtains an optimal solution in $O\left(r^2\right)$ time, where r is the number of initial operations. Their second algorithm, which is based on a refinement of their initial algorithm, obtains an optimal solution in $O\left(r\left(\min\left\{r, m^2\right\} + m\log n\right)\right)$ time, which improves the time of their original algorithm in problem instances where $m \leq r/\log n$. A seminal algorithm utilizing matrices and decrementing sets, based on the *Longest Alternate Processing Time first* (LAPT) rule, provided by Pinedo (2012) for the non-preemptive version of this problem denoted by $Om \mid \mid C_{\max}$, is shown to obtain optimal schedules for the two-machine preemptive open shop problem $O2 \left|pmtn\right| C_{\max}$.

The solution for problem $Om \left|pmtn\right| C_{\max}$ is presented next. The problem is solved by finding a decomposition of a Δ-doubly stochastic matrix into the weighted sum of permutation matrices, followed by assembling an optimal schedule based on that decomposition. Before we proceed, it is essential that we define the following:

Definition 2.3. *A square $(h \times h)$ matrix $A = (\alpha_{i,j})$ is called* ***doubly stochastic*** *if every $\alpha_{i,j}$ in the matrix satisfies the following requirements:*

$$\begin{aligned} 0 &\leq \alpha_{i,j} \leq 1 \\ \sum_{i=1}^{h} \alpha_{i,j} &= 1,\ 1 \leq j \leq h \\ \sum_{j=1}^{h} \alpha_{i,j} &= 1,\ 1 \leq i \leq h \end{aligned}$$

Definition 2.4. *A doubly stochastic matrix is called a* ***permutation matrix*** *if $\alpha_{i,j} \in \{0,1\}$, for every $1 \leq j \leq h$ and $1 \leq i \leq h$.*

Definition 2.5. *A* ***line*** *describes either a row or a column of matrix. For a square $(h \times h)$-matrix, a* ***diagonal*** *describes a set of elements, which do not belong to the same line.*

Next we outline the theorem which was described initially by Birkhoff (1946) and discovered independently by von Neumann (1953), where they show that a Δ-doubly stochastic matrix can be decomposed into a linear combination of permutation matrices. This theorem is particularly important as it directly implies that the problem of minimizing a linear function defined for a set of elements, can be solved as a linear programming problem.

Theorem 2.3 (Birkhoff (1946), von Neumann (1953))**.** *It is possible to decompose a Δ-doubly stochastic matrix A into a linear combination of permutation matrices*

$\Pi_1, \Pi_2, \ldots, \Pi_k$ *such that*

$$A = \sum_{k=1}^{u} \delta_k \Pi_k,$$

$$\sum_{k=1}^{u} \delta_k = \Delta,$$

$$0 \leq \delta_k \leq \Delta, \ 1 \leq k \leq u.$$

By considering the Birkhoff-von Neumann theorem, as well as the above definitions, the solution to $O\left|pmtn\right|C_{\max}$ is described. Initially the original processing times are placed into an $(m \times n)$ matrix $P = (p_{ij})$. The value of Δ is computed such that

$$\Delta = \max\left\{\max\left\{\sum_{i=1}^{m} p_{ij} | j \in n\right\}, \max\left\{\sum_{j=1}^{n} p_{ij} | 1 \leq i \leq m\right\}\right\}.$$

Define $h = m+n$, and construct a Δ-doubly stochastic matrix A of the following form:

$$A = \begin{pmatrix} P_{m\times n} & D_m \\ D_n & B_{n\times m} \end{pmatrix},$$

where D_m and D_n are $(m \times m)$ and $(n \times n)$ diagonal matrices, with the value of each off-diagonal element equal to zero. The diagonal elements are chosen in order to provide the total sum of each m first rows, and of each n first columns of matrix A equal to Δ. Matrix B can be found by subsequently making lines of matrix A have sums equal to Δ.

The following algorithm finds a desired decomposition of a Δ-doubly stochastic matrix:

Algorithm Birkhoff-von Neumann

INPUT: A Δ-doubly stochastic matrix A.

OUTPUT: A decomposition of matrix A.

Step 1. Define $k = 1$.

Step 2. Find diagonal $R(A)$ without zero elements. Let δ be the smallest element of $R(A)$ .Define $\delta_k = \delta$. Find permutation matrix $\Pi_k = \left(\pi_{i,j}^k\right)$ where $\pi_{i,j}^k = 1$ if $\alpha_{i,j}$ belongs to $R(A)$.

Step 3. Subtract δ from each element of the diagonal $R(A)$, call the obtained matrix A again. If A is not the zero matrix, then it is Δ'-doubly stochastic, where

$\Delta' \leq \Delta$, define $k = k + 1$ and repeat Step 2. Otherwise if matrix A is zero, stop.

Following the above decomposition of matrix A, we can proceed with the construction of an optimal schedule for the original $O\,|pmtn|\,C_{\max}$ problem. This is achieved by splitting the time interval $[0, \Delta]$ into a number of u sub intervals, where in each of these sub intervals, jobs are processed on each of the machines.

Algorithm O-Match

INPUT: A linear decomposition of matrix A

OUTPUT: An optimal preemptive schedule for problem $O\,|pmtn|\,C_{\max}$.

Step 1. Split time interval $[0, \Delta]$ into u sub-intervals $I_1, I_2, \ldots, I_u$, such that

$$\begin{aligned} I_1 &= [0, \delta_1], \\ I_2 &= [\delta_1, \delta_1 + \delta_2], \\ &\ldots, \\ I_u &= \left[\sum_{k=1}^{u-1} \delta_k, \sum_{k=1}^{u} \delta_k\right], \end{aligned}$$

where

$$\sum_{k=1}^{u} \delta_k = \Delta.$$

Step 2. Starting from $k = 1$ to $k = u$, for permutation matrix $\Pi_k = \left(\pi_{i,j}^k\right)$, if $\pi_{i,j}^k = 1$ for $i \leq m$ and $j \leq n$, then job $J_j,\ 1 \leq j \leq n$, is processed on machine M_i during the time interval I_k.

The running time of this algorithm depends on the time required for finding a diagonal, which can be done in polynomial time, as the problem of finding a diagonal is related to the assignment problem described in Section 1.2.

2.3.2 Minimizing Makespan

Consider the problem of minimizing the makespan on unrelated machines. In this case, the processing time of each job is machine dependent, thus is represented by a matrix $P = (p_{ij})_{m \times n}$. When preemption is allowed, jobs may be processed by multiple machines, subject to the consideration that no job is processed by more than one machine at a time. Furthermore, for a preemptive schedule S_p, let t_{ij} represent the amount of time a job J_j, $1 \leq j \leq n$, is processed by machine M_i, so that if j has fulfilled its processing requirement, then

$$\sum_{i=1}^{m} \frac{t_{ij}}{p_{ij}} = 1.$$

Finding an optimal schedule for the preemptive version of this problem, can be achieved in polynomial time. This is demonstrated in a two-stage solution to $Rm\,|pmtn|\,C_{\max}$, due to Lawler and Labetoulle (1987), which consists of solving two polynomially solvable problems. In the first stage of the solution, Lawler and Labetoulle (1987) find the optimal value for the makespan by solving the following linear programming formulation of the problem.

$$\begin{aligned}
&\text{Minimize} \quad C \\
&\text{Subject to:} \\
&\qquad \textstyle\sum_{i=1}^{m} \frac{t_{ij}}{p_{ij}} = 1, \quad \text{for} \quad 1 \leq j \leq n, \\
&\qquad \textstyle\sum_{j=1}^{n} t_{ij} \leq C, \quad \text{for} \quad 1 \leq i \leq m \\
&\qquad \textstyle\sum_{i=1}^{m} t_{ij} \leq C, \quad \text{for} \quad 1 \leq j \leq n \\
&\qquad t_{ij} \geq 0 \qquad\quad \text{for} \quad 1 \leq j \leq n, 1 \leq i \leq m
\end{aligned} \tag{2.20}$$

In the above LP formulation, C corresponds to the value of the makespan. The first set of constraints in (2.20) ensure the completion of all jobs, while the second set of constraints ensure that all machines will complete jobs assigned to them before time C.

Let C^* and t_{ij}^* be the corresponding values of the variables, obtained by solving (2.20). Then there is an optimal schedule S_p^*, so that $C_{\max}\left(S_p^*\right) = C^*$. As C^* is minimal, we obtain

$$C^* = \max\left\{\max\left\{\sum_{i=1}^{m} t_{ij}^* \mid 1 \leq j \leq n\right\}, \max\left\{\sum_{j=1}^{n} t_{ij}^* \mid 1 \leq i \leq m\right\}\right\}. \tag{2.21}$$

It is apparent that (2.21) closely resembles the value of the makespan for problem $Om\,|pmtn|\,C_{\max}$ given in (2.19). In the second stage of the solution, Lawler and Labetoulle (1987) find an optimal schedule by solving $Om\,|pmtn|\,C_{\max}$ with processing times t^*_{ij}.

The non-preemptive version of this problem, denoted by $Rm\,|\,|\,C_{\max}$, is a generalization of the problem of finding a non-preemptive schedule on uniform machines, and as a result is NP-hard. The computationally hard nature of this problem has prohibited the development of polynomial time algorithms which can solve this problem to optimality, this has lead researchers in the past several decades to develop a variety of approximation methodologies for obtaining near-optimal results for this problem. For this problem, a list scheduling-based algorithm by Davis and Jaffe (1981) was shown to produce a schedule within $2\sqrt{m}$ the optimum makespan. For the problem on two unrelated machines, Potts (1985) shows that a heuristic based on linear programming and enumeration obtains surprisingly near optimal results. The process of obtaining a non-preemptive schedule with the Linear Programming and Enumeration (LPE) heuristic described by Potts, consists of two stages: Initially a linear programming problem is solved in order to obtain a partial schedule, then brute-force enumeration is used to construct the remaining schedule. The problem of minimizing the makespan for $Rm\,||\,C_{\max}$ is defined in the following integer programming formulation due to Potts (1985):

$$\begin{array}{lll}
\text{Minimize} & C_{\max} & \\
 & & \\
\text{Subject to:} & & \\
 & \sum_{j=1}^{n} p_{ij}x_{ij} \le C_{\max} & ,\text{ for } \; 1 \le i \le m, \\
 & \sum_{i=1}^{m} x_{ij} = 1 & ,\text{ for } \; 1 \le j \le n, \\
 & x_{ij} \in \{0,1\} & ,\text{ for } \; 1 \le i \le m,\ 1 \le j \le n.
\end{array}$$

For the first stage of the solution, Potts considers a relaxation of the integrality constraint for the decision making variable. Thus, by considering $0 \le x_{ij} \le 1$, the above IP formulation is effectively transformed into a linear programing problem, which is provided next:

$$
\begin{aligned}
&\text{Minimize} \quad C_{\max} \\
&\text{Subject to:} \\
&\qquad \textstyle\sum_{j=1}^{n} p_{ij}x_{ij} \leq C_{\max} && \text{, for} \quad 1 \leq i \leq m, \\
&\qquad \textstyle\sum_{i=1}^{m} x_{ij} = 1 && \text{, for} \quad 1 \leq j \leq n, \\
&\qquad x_{ij} \geq 0 && \text{, for} \quad 1 \leq i \leq m, \ 1 \leq j \leq n, \\
&\qquad x_{ij} \leq 1 && \text{, for} \quad 1 \leq i \leq m, \ 1 \leq j \leq n.
\end{aligned} \tag{2.22}
$$

The above formulation closely resembles that of Lawler and Labetoulle for the preemptive version of the makespan problem on unrelated machines in (2.20), where $t_{ij} = p_{ij}x_{ij}$, but with the essential difference that there is no constraint limiting the execution of jobs by one processor at a time. Following this LP formulation, a partial schedule is obtained by observing the value of the assignment variable: For each job J_j, $1 \leq j \leq n$, if the value of x_{ij} is equal to 1, then j is scheduled on machine M_i. Jobs which do not satisfy this condition, i.e. they have partial assignments to multiple machines, are called *fractional jobs*. Potts further shows that for any instance of this problem where $n \geq m - 1$, there are at most $m - 1$ fractional jobs.

In the second stage of the solution, the final schedule is obtained by appending fractional jobs to the end of the final schedule. The final assignment of these fractional jobs is determined by enumerating all possible assignments. The LPE heuristic runs in polynomial time, as in the second stage of the solution, a maximum of m^{m-1} schedules are generated and compared, while in the first stage, LP problems also require polynomial time. For the problem with two unrelated machines, $R2 \,|\, | C_{\max}$, LPE finds a schedule S^{LPE} in $O(n)$ time, so that its worst-case performance ratio is given by the inequality

$$\frac{C_{\max}\left(S^{LPE}\right)}{C_{\max}\left(S^{*}\right)} \leq \frac{1+\sqrt{5}}{2},$$

and is tight for an instance with two jobs, where for some $p > 0$,

$$\begin{aligned} p_{1,1} &= \frac{\left(-1+\sqrt{5}\right)p}{2}, \\ p_{1,2} &= p, \\ p_{2,1} &= p, \\ p_{2,2} &= \frac{\left(1+\sqrt{5}\right)}{2}. \end{aligned}$$

Furthermore, Potts proves that for the general case with m machines, where $m \geq 3$, the worst-case performance ratio of LPE is

$$\frac{C_{\max}\left(S^{LPE}\right)}{C_{\max}\left(S^{*}\right)} \leq 2.$$

In the same paper, Potts also provides the LPE$'$ heuristic: A modified version of LPE, where having obtained a fractional job J_k after an initial application of LPE, the heuristic is then applied to a modified version of the problem which has the constraint that J_k is assigned to the machine on which it has the smallest processing time. Finally the better of the two schedules produced by each application of LPE is selected. Heuristic LPE' finds a schedule in $O(n)$ time, and its worst case ratio is given by

$$\frac{C_{\max}\left(S^{LPE'}\right)}{C_{\max}\left(S^{*}\right)} \leq \frac{3}{2}.$$

Following the same principle of linear programming relaxation of integer programming problems, Lenstra, Shmoys and Tardos (1990) show that the fractional solution obtained from (2.22), can be rounded to a satisfactory integral approximation in polynomial time. In essence, this removes the need for enumeration in the LPE heuristic, thus removing the exponential dependence on the number of machines. The results by Lenstra, Shmoys and Tardos (1990) are based on a rounding technique which is presented next: Consider a set of jobs $N_i(t)$ that require to be processed for no more than t amount of time on machine M_i, and a set $\mathcal{M}_j(t)$ of machines that can process job J_j, $1 \leq j \leq n$, in no more than t time. Furthermore each machine is associated with a deadline d_i and each job is constrained to be processed for a maximum of t amount of time on every machine. Thus the feasibility of a schedule is determined by simply checking if the criteria of the problem have been satisfied.

Theorem 2.4 (Lenstra, Shmoys and Tardos (1990))**.** *For a matrix of processing times $P = (p_{ij})_{m\times n,}$, a vector of machine deadlines $\vec{d} = (d_1, d_2, \ldots, d_m)$, where D represents the maximum deadline, so that $D = \max\{d_i \mid 1 \leq i \leq m\}$, and a time duration t, if* *the linear program* $LP\left(P,\vec{d},t\right)$, *given by*

$$\begin{array}{ll} \textit{Minimize} & D \\ \textit{Subject to:} & \\ & \sum_{j=1}^{n} p_{ij}x_{ij} \le d_i \quad \textit{, for} \quad 1 \le i \le m,\ 1 \le j \le n_i(t) \\ & \sum_{i=1}^{m} x_{ij} = 1 \qquad \textit{, for} \quad 1 \le j \le n,\ m \in \mathcal{M}_j(t) \\ & x_{ij} \ge 0 \qquad \textit{, for} \quad 1 \le i \le m,\ 1 \le j \le n_i(t), \end{array} \tag{2.23}$$

has a feasible solution, then any vertex $\tilde{x}$ *of this polytope can be rounded to a feasible solution* $\bar{x}$ *of the integer program* $IP\left(P,\vec{d},t\right)$, *given by*

$$\begin{array}{ll} \textit{Minimize} & D \\ \textit{Subject to:} & \\ & \sum_{j=1}^{n} p_{ij}x_{ij} \le d_i + 1 \quad \textit{, for} \quad 1 \le i \le m, \quad 1 \le j \le n_i(t) \\ & \sum_{i=1}^{m} x_{ij} = 1 \qquad \textit{, for} \quad m \in \mathcal{M}_j(t),\ 1 \le j \le n, \\ & x_{ij} \in \{0,1\} \qquad \textit{, for} \quad 1 \le i \le m,\ 1 \le j \le n_i(t), \end{array}$$

and this rounding can be done in polynomial time.

Algorithm R-ROUND (Lenstra, Shmoys and Tardos (1990))

INPUT: A problem instance for $Rm\,||\,C_{\max}$

OUTPUT: An approximate schedule S_{np} for this problem.

Step 1. Solve (2.23) and obtain a feasible solution vertex $\tilde{x}$.

Step 2. For each job J_j, $1 \le j \le n$, and machine M_i, $1 \le i \le m$, if $x_{ij} = 1$, assign j to M_i.

Step 3. Define a set of fractional jobs $F \subset N$, such for every $j \in F$, $0 < x_{ij} < 1$ and $1 \le i \le m$. Construct bipartite graph G' on vertex set $F \cup \mathcal{M}$.

Step 4. Find a *maximum matching* $\mathcal{N}$ in H.

Step 5. Assign fractional jobs to machines according to matching $\mathcal{N}$.

Step 6. Stop.

The R-ROUND procedure finds a non-preemptive schedule S_{np} with makespan arbitrarily close to twice the optimum, so that

$$\frac{C_{\max}\left(S_{np}\right)}{C_{\max}\left(S^{*}\right)} \leq 2.$$

2.3.3 Minimizing Sum of Completion Times

Following the trend on identical and uniform machines for the problem of minimizing the total completion times when machines are unrelated, is polynomially solvable. The polynomial time solvability of this problem, denoted by $Rm \,|\,|\sum C_j$, is demonstrated by Brucker (2007), where this problem is reduced to an assignment problem. Similarly to the problem with uniform machines, if in an optimal schedule machine M_i has been assigned r jobs in the sequence $I_i := \{j_1, j_2, \ldots, j_r\}$, where $1 \leq j \leq n$, then the contribution of each job in I_i to the makespan is given by:

$$rp_{ij} + (r-1)\, p_{ij_2} + \ldots + p_{ij_r}.$$

The numbering of position in which jobs are processed on a machine, is given in a non-increasing order, such that the first job to be processed on machine M_i is assigned to the r-th position (from the end of the schedule) and the last job to be processed by M_i is assigned to position 1. The problem is subsequently solved as an assignment problem, where each job $J_j,\ 1 \leq j \leq n$, has to be assigned to a position on one of the m machines. Optimal solutions from this assignment problem, are associated with the following property: If a job is assigned to a position $k > 1$ on machine M_i, then there is a job assigned to position $k-1$ on the same machine. Alternatively scheduling job $J_j,\ 1 \leq j \leq n$, in position $k-1$ would improve the total cost of the assignment. Let x_{ikj} denote a 0-1 decision making variable, so that $x_{ikj} = 1$ if job $J_j,\ 1 \leq j \leq n$, is scheduled k-th to last on machine M_i and $x_{ikj} = 0$ otherwise. The problem can then be formulated as an integer programming problem. This is given next.

$$\text{Minimize} \quad \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{n} kp_{ij}x_{ikj}$$

Subject to:

$$\sum_{i=1}^{m}\sum_{k=1}^{n} x_{ikj} = 1 \qquad \text{, for} \quad 1 \leq j \leq n,$$

$$\sum_{j=1}^{n} x_{ikj} \leq 1 \qquad \text{, for} \quad 1 \leq i \leq m,\ k \in N$$

$$x_{ikj} \in \{0,1\} \qquad \text{, for} \quad 1 \leq i \leq m,\ 1 \leq j \leq n,\ k \in N$$

This formulation corresponds to the WEIGHTED MATCHING PROBLEM in bipartite graphs, where each of the n jobs has to be matched to one of nm positions, as each machine may process a total of n jobs. Each matching of a job J_j, $1 \leq j \leq n$, to a position ik is associated with a cost kp_{ij}. The objective is to obtain a matching which minimizes the total cost. This is a classical problem of combinatorial optimisation, which is called the minimum weight matching problem, that is known to be solvable in polynomial time.

An interesting problem arises when preemption is allowed. The preemptive version of the total flow time problem on unrelated machines, denoted by $Rm\,|pmtn|\sum C_j$, is known to be strongly NP-hard due to Sitters (2005), as this problem has a polynomial reduction from the 3-DIMENTIONAL MATCHING PROBLEM, which is known to be strongly NP-hard due to Karp (1972). The complexity classification of this preemptive problem is a considerable contrast to other preemptive scheduling models, as it is the only known classical machine scheduling problem where the introduction of preemption actually *increases* the complexity of the problem. Furthermore, Sitters (2008) shows that the problem is also APX-hard, thus no PTAS may be obtained for this problem, unless $P = NP$.

CHAPTER 3

Power of Preemption

In the previous chapter we presented a number of scheduling problems on parallel machines. As one may observe, a large number of problems benefit from the introduction of preemption, as preemptive problems often benefit from reduced complexity, and reduce the overall value of objective functions.

Consider an instance of a scheduling problem to minimize an objective function Φ on m parallel machines. For the corresponding problem, the power of preemption is defined as the maximum ratio $\Phi\left(S_{np}^{*}\right)/\Phi\left(S_{p}^{*}\right)$ across all instances of the problem. The power of preemption is denoted by ρ_m. The power of preemption determines what can be gained regarding the maximum completion time if preemption is allowed.

In order to determine the exact value of ρ_m for a given problem, and to give that concept some practical meaning,the following should be done:

(i) demonstrate that the inequality

$$\frac{\Phi\left(S_{np}^{*}\right)}{\Phi\left(S_{p}^{*}\right)} \leq \rho_m \tag{3.1}$$

holds for all instances of a given problem;

(ii) demonstrate that the value of the of ρ_m is tight by exhibiting instances for which (3.1) holds as an equality; and

(iii) for problems which are NP-hard, develop a polynomial-time algorithm that finds a heuristic non-preemptive schedule S_{np}, such that

$$\frac{\Phi\left(S_{np}^{*}\right)}{\Phi\left(S_{p}^{*}\right)} \leq \frac{\Phi\left(S_{np}\right)}{\Phi\left(S_{p}^{*}\right)} \leq \rho_m.$$

3.1 Power of Preemption on Identical Machines

When machines are identical, consider the power of preemption for the problem of minimizing the makespan, so that for some schedule S, $\Phi(S) = C_{\max}(S)$. Thus for this problem the power of preemption is given by

$$\frac{C_{\max}\left(S^*_{np}\right)}{C_{\max}\left(S^*_{p}\right)} \leq \rho_m.$$

As shown in Section 2.1, an optimal preemptive schedule S^*_p is obtained via McNaughton's wrap-around algorithm in linear time. Furthermore the value of the optimal preemptive makespan is given by (2.7). For this problem, only instances where there are no less jobs than machines are considered, as for the case $n < m$, only the problem with n machines needs to be considered.

In Section 2.1.1, two classes of instances are identified, depending on where the maximum is achieved on the right-hand side of (2.7); see Definition 2.1.

For the Class 2 instances, where the makespan of the preemptive schedule is given by the average machine load T given in (2.5), i.e. $C_{\max}\left(S^*_p\right) = T$, the upper bound on the power of preemption is known to be

$$\rho_m \leq 2 - \frac{2}{m+1}, \tag{3.2}$$

as proved by Braun and Schmidt (2003) and independently by Lee and Strusevich (2005).

Braun and Schmidt (2003) show that a non-preemptive schedule which satisfies (3.2) can be obtained in $O(n \log n)$ time by the application of the LPT list scheduling algorithm described in Section 2.1, while the results by Lee and Strusevich (2005) indicate that a schedule which satisfies the bound can be obtained in $O(n)$ time.

Indeed, Lee and Strusevich (2005) describe a class of heuristic algorithms that find a non-preemptive schedule S_{np} for Class 2 instances of the problem, such that (3.2) holds. Specifically, a heuristic algorithm is said to belong to Class AP if it creates a schedule with the following properties:

(i) jobs are split into m subsets $N_1, N_2, \ldots N_m$, so that jobs in set N_i are scheduled on machine M_i;

(ii) no machine is idle in the time interval $[0, p(N_i)]$;

(iii) suppose that machine M_1 terminates the schedule, and job J_k is the last job assigned to that machine. Then the starting time of job J_k is smaller that $\min \{p(N_i) \,|2 \le i \le m\}$;

(iv) for the terminating job J_k, the inequality $p_k \le \min \{p(N_i) \,|2 \le i \le m\}$ holds.

Based on the above description, Lee and Strusevich (2005) prove that any schedule obtained by an AP-class algorithm satisfies (3.2). The proof uses the fact that if the total processing time of a subset of jobs N_i exceeds (3.2), then the terminating job J_k can be transferred to a different machine, or can be swapped with a different subset of jobs, resulting in a schedule with a smaller makespan which satisfies (3.2).

The tightness of (3.2) can be demonstrated by considering an instance with $m + 1$ unit length jobs. In this case an optimal non-preemptive schedule is obtained by scheduling two jobs on one of the machines, while the remaining machines receive one job each, so that $C_{\max}\left(S^*_{np}\right) = 2$. The average machine load for this instance is given by

$$T = \frac{m+1}{m}, \tag{3.3}$$

which satisfies the inequality $p_1 < T$, hence the makespan of the preemptive schedule is given by (3.3). Thus, it is apparent that for this instance the power of preemption is given by

$$\begin{aligned} \rho_m &\le \frac{2}{\frac{m+1}{m}} \\ &= 2\frac{m}{m+1} \end{aligned}$$

which satisfies (3.2).

Define g, $0 \le g \le m - 1$, as the smallest integer such that

$$p_{g+1} \le \frac{p(N) - \sum_{j=1}^{g} p_j}{m-g}. \tag{3.4}$$

Notice that for the Class 2 instances $g = 0$ and for the Class 1 instances $g \ge 1$.

For Class 1 of instances, where the makespan of the optimal preemptive schedule is defined by the processing time of the longest job, so that $C_{\max}\left(S^*_p\right) = p_1$, Rustogi and Strusevich (2013) show that

$$\rho_m \le \left(2 - \frac{2}{m-g+1}\right). \tag{3.5}$$

Rustogi and Strusevich (2013) further provide an algorithm which finds a non-preemptive schedule which satisfies either of bounds (3.2) and (3.5), depending on the input instance. This algorithm is described next.

Algorithm P-nonpreemptive [Rustogi and Strusevich (2013)]

INPUT: An instance of problem $Pm\,|\,|\,C_{\max}$

OUTPUT: A non-preemptive schedule S^{*}_{np}, which satisfies either bound (3.2) or (3.5).

Step 1. Find the m largest jobs and sort them in LPT order.

Step 2. Determine index g, $0 \leq g \leq 1$, as the smallest index which satisfies (3.4).

Step 3. Obtain a partial schedule by assigning each job J_k, $1 \leq k \leq g$ to machine M_k.

Step 4. Complete the partial schedule by applying Algorithm LS (as described in Section 2.1) to the remaining jobs and machines in the schedule.

This algorithm runs in $O\left(nm\right)$ time, provided that the LPT sequence of jobs is known.

In order to demonstrate the tightness of (3.5), Rustogi and Strusevich (2013) consider an instance with $m+1$ jobs, where there are $g \geq 1$ jobs with a larger processing time of length $m-g+1$, and $n-g$ jobs with the processing time $m-g$. Thus, the total processing time is given by $p\left(N\right) = g\left(m-g+1\right) + \left(m-g+1\right)\left(m-g\right) = m\left(m-g+1\right)$, where $p_1 = \left(m-g+1\right)$. In this case, each of the g longer jobs is assigned to an individual machine, while the remaining $\left(m-g\right)+1$ jobs are processed on the remaining $m-g$ machines, so that on some machine M_l, $g+1 \leq l \leq m$, two jobs will be processed, with the remaining machines processing exactly one job each. The makespan of the optimal non-preemptive schedule for this instance is given by $C_{\max}\left(S^{*}_{np}\right) = 2m-2g$, while the optimal preemptive schedule has length $C_{\max}\left(S^{*}_{p}\right) = m-g+1$, thus $\rho_m = \left(2m-2g\right)/\left(m-g+1\right)$, which satisfies (3.5).

3.2 Power of Preemption on Uniform Machines

In this section, we review the results on determining the power of preemption on uniform parallel machines.

3.2.1 Makespan

We begin with presenting the latest results by Soper and Strusevich (2014b) for the power of preemption for the general case of minimizing the makespan on m uniform machines.

As described in Section 2.2.1, the makespan of an optimal preemptive schedule is given by (2.16), which is obtained in $O(n + m \ \log m)$ time by an application of the algorithm due to Gonzalez and Sahni (1978).

Recall that the instances of the problem are classified in accordance with Definition 2.2. Notice that an instance may belong to several classes simultaneously, if there is a tie for the maximum value of T_u.

Consider a set of Class m instances, for which the makespan of the preemptive schedule is given by T_m, as defined in (2.18). Woeginger (2000) shows that for instances of this type, Algorithm QLPT finds a non-preemptive schedule S_{np}, so that the bound

$$\rho_m \le 2 - \frac{1}{m}, \tag{3.6}$$

holds and is tight.

Initially, consider the following definitions due to Soper and Strusevich (2014b).

Definition 3.1 (Soper and Strusevich (2014b))**.** *For an instance $I = (\mathcal{L}_n, \mathcal{M}_m)$, suppose that in a non-preemptive schedule $S_{np}(I)$ the last completed operation is that of processing job J_h, $1 \le h \le n$, on machine M_k, $1 \le k \le m$. Job J_h is called the terminal job, and machine M_k the critical machine.*

Definition 3.2 (Soper and Strusevich (2014b))**.** *For the problem of minimizing the makespan on m uniform machines, an instance I is called canonical if for each machine M_k there exists an optimal non-preemptive schedule such that M_k is the only critical machine.*

For $n \ge m$, $\mathcal{I}$ represents the set of instances $I = (\mathcal{L}_n, \mathcal{M}_m)$, such that

- All jobs have equal processing times, i.e. $p_j = p$, $1 \le j \le n$;
- The speeds of the machines are positive integers that for a positive W satisfy (2.15), so that

$$s_1 \ge s_2 \ge \ldots \ge s_m; \ 1 \le W s_i \le m; \ \sum_{i=1}^{m} s_i = \frac{n+m-1}{W}.$$

In the following theorem, Soper and Strusevich (2014b) provide the necessary and sufficient conditions for tight instances of Class m.

Theorem 3.1 (Soper and Strusevich (2014b))**.** *For an instance $I = (\mathcal{L}_n, \mathcal{M}_m)$ of Class m to be tight, it is necessary and sufficient that I is an instance of set $\mathcal{I}$ with $n = m$.*

Non-preemptive schedules for Class m instances, for which (3.6) holds, are obtained via a modified version of the LPT list scheduling algorithm. This is described next.

Algorithm LPTm (Soper and Strusevich (2014b))

INPUT: A Class m instance $I = (\mathcal{L}_n, \mathcal{M}_m)$ for problem $Qm \,|\,| C_{\max}$.

OUTPUT: A non-preemptive schedule $S_{LPT}(I)$.

Step 1. If required, renumber the jobs so that the m longest jobs are numbered in accordance with (2.1), while the other jobs are numbered arbitrarily.

Step 2. At any time that a machine becomes available, take the first job in the current list $\mathcal{L}_n$ and assign it to the machine on which it will complete as early as possible. Remove the assigned job from the list.

Step 3. Repeat Step 2 until all jobs are assigned.

Compared to the full version of the LPT algorithm, the modified Algorithm LPTm requires only $O(m \log m + nm)$ time, as finding and sorting the m longest jobs requires $O(m \log m)$ time.

Although bound (3.6) holds for all instances, irrespective of their class, the upper bound can be reduced for instances that are known to belong to Class r, $1 \le r \le m-1$. Soper and Strusevich (2014b) prove that the bound

$$\rho_m \le \max\left\{2 - \frac{1}{r}, 2 - \frac{1}{m-r}\right\}, \tag{3.7}$$

for the power of preemption, holds for any arbitrary instance I of Class r, $1 \le r \le m-1$. If r is not unique, the value closest to $m/2$ is selected.

The corresponding non-preemtive schedule can be found by an algorithm presented below. For r, $1 \le r \le m-1$, lists $\mathcal{L}_r^{'}$ and $\mathcal{M}_r^{'}$ are obtained by the removal of the r longest jobs and r fastest machines from lists $\mathcal{L}_n$ and $\mathcal{M}_m$ respectively, so that $\mathcal{L}_r^{'} := \{p_{r+1}, \dots, p_n\}$ and $\mathcal{M}_r^{'} := \{M_{r+1}, \dots, M_m\}$. A non-preemptive schedule for a Class r, $1 \le r \le m-1$, instance $I = (\mathcal{L}_n, \mathcal{M}_m)$ for which bound (3.7) holds, is obtained via the following algorithm, in which Algorithm LPTm is applied to each of the instances $I_r = (\mathcal{L}_r, \mathcal{M}_r)$, and $I_r' = \left(\mathcal{L}_r', \mathcal{M}_r'\right)$.

Algorithm LPTr (Soper and Strusevich (2014b))

INPUT: A Class m instance $I = (\mathcal{L}_n, \mathcal{M}_m)$ for problem $Qm \mid \mid C_{\max}$.

OUTPUT: A non-preemptive schedule $S_{LPT(r)}(I)$.

Step 1. Split Class r instance $I = (\mathcal{L}_n, \mathcal{M}_m)$ into two instances $I_r = (\mathcal{L}_r, \mathcal{M}_r)$, and $I_r' = (\mathcal{L}_r', \mathcal{M}_r')$.

Step 2. Run algorithm LPTm for I_r, and I_r', to find schedules $S_{LPT}(I_r)$ and $S_{LPT}(I_r')$.

Step 3. Output Schedule $S_{LPT(r)}(I)$, obtained by combining schedules $S_{LPT}(I_r)$ and $S_{LPT}(I_r')$.

Algorithm LPTr finds the required schedule in $O(m \log m + nm)$ time.

Consider an instance I for which bound (3.7) is tight. It can be simply observed that there are two cases for which I is a tight instance, depending on where the maximum is achieved in (3.7). For the case

$$\rho_m = 2 - \frac{1}{r}, \tag{3.8}$$

the instance which delivers this bound is given in the following lemma.

Lemma 3.1 (Soper and Strusevich (2014b)). *For $n \geq m$, and r such that $1 \leq r \leq m-1$ and $2r \geq m$, there exists an instance $I = (\mathcal{L}_n, \mathcal{M}_m)$ of Class r, such that (3.8) holds.*

Proof: For a given m, take an arbitrary r, such that $1 \leq r \leq m-1$ and $2r \geq m$. Consider an instance I of Class r with m machines, and $n = m$ jobs. The $r-1$ faster machines each have speed 2 so that $s_i = 2$, $1 \leq i \leq r-1$, while the remaining machines have unit speeds, i.e. $s_i = 1$, $r \leq i \leq m$. Furthermore the r longest jobs each have processing time $p_j = 1$, $1 \leq j \leq r$, while the remaining jobs have processing time $p_j = r/(2r-1) < 1$, $r+1 \leq j \leq m$. This yields

$$T_i = \frac{1}{2}, 1 \leq i \leq r-1;$$

$$T_r = \frac{r}{2r-1} > \frac{1}{2};$$

and

$$T_i = \frac{r}{2r-1},\ r+1 \le i \le m.$$

Here, $T_r = T_{r+1} = \ldots = T_m$, and r is the index closest to $m/2$ due to $r \ge m/2$. Thus, instance I belongs to Class r and the makespan of the preemptive schedule is given by

$$C_{\max}\left(S_p^*(I)\right) = \frac{r}{2r-1}.$$

For makespan of an optimal non-preemptive schedule, it holds that

$$C_{\max}\left(S_{np}^*(I)\right) = 1.$$

Thus,

$$\frac{C_{\max}\left(S_{np}^*(I)\right)}{C_{\max}\left(S_p^*(I)\right)} = 2 - \frac{1}{m-r},$$

which satisfies (3.8). □

The next lemma demonstrates a tight instance for the case

$$\rho_m = 2 - \frac{1}{m-r}. \tag{3.9}$$

Lemma 3.2 (Soper and Strusevich (2014b))**.** *For $n \ge m$, and r such that $1 \le r \le m-1$ and $2r < m$, there exists an instance I of Class r such that (3.9) holds.*

Proof: For a given m, take an arbitrary r, such that $1 \le r \le m-1$ and $2r < m$. Consider an instance I of Class r with $m = n$ jobs. In this instance, the $m-1$ fastest machines have speed equal to 2, so that $s_i = 2$, $1 \le i \le m-1$, while the m-th machine has speed $s_m = 1$. For a value V such that

$$V = \frac{m-r}{2(m-r)-1},$$

the processing times of the r longest jobs is given by $p_j = 2Q$, $1 \le j \le r$, with the remaining jobs having unit length time, so that $p_j = 1$, $r+1 \le j \le m$. This yields

$$\begin{array}{cc} T_i = V, & 1 \le i \le r; \\ T_i = \frac{2rV+(i-r)}{2r+2(i-r)}, & r+1 \le i \le m; \\ T_m = V. & \end{array}$$

In this case, $T_1 = T_2 = \ldots = T_r$, where r is the index closest to $m/2$ due to $r < m/2$.

Thus I is a Class r instance and

$$C_{\max}\left(S_p^*\left(I\right)\right) = V.$$

An optimal non-preemptive schedule for instances of this type is obtained by assigning exactly one of the longer jobs to each of the $m-1$ faster machines, and the m-th job is assigned to the slower machine M_m. The makespan of this schedule is given by

$$C_{\max}\left(S_{np}^*\left(I\right)\right) = 1.$$

Thus,

$$\frac{C_{\max}\left(S_{np}^*\left(I\right)\right)}{C_{\max}\left(S_p^*\left(I\right)\right)} = 2 - \frac{1}{m-r},$$

which satisfies (3.9). □

3.2.2 Makespan for a Fixed Number of Uniform Machines

For problems on 2 and 3 uniform machines, Soper and Strusevich (2014a) examine the behaviour of the power of preemption as the speed of the fastest machine increases, this parametric analysis is performed with respect to the speed of the fastest machine, for cases where machine speeds can take either of two values.

For problem $Q2 \,|\, \,|\, C_{\max}$, Soper and Strusevich (2014a) distinguish between two classes of instances for this problem in accordance with Definition 2.2, depending on the expression which gives the value of the makespan for this problem. Those are:

- Class 1, for which $C_{\max}(S_p^*) = p_1/s$;
- Class 2, for which $C_{\max}(S_p^*) = T_2$, where

$$T_2 = \frac{p\left(N\right)}{s+1} \tag{3.10}$$

For a Class 1 instance, consider a schedule S_{np} defined by the sets $N_1 = \{J_1\}$ and $N_2 = N \backslash \{J_1\}$. Due to $\frac{p_1}{s} > p(N) - p_1$, it is simple to show that $C_{\max}(S_{np}) = C_{\max}(S_p^*) = p_1/s$, so that allowing preemption would not reduce the makespan. Due to this reason, only Class 2 instances of the problem are considered. Soper and Strusevich (2014a) provide an algorithm which finds a non-preemptive schedule S_{np} for which the upper bound

$$\rho_2 \leq \frac{C_{\max}(S_{np})}{C_{\max}(S_p^*)} \leq \Phi(s), \tag{3.11}$$

holds, where

$$\Phi(s) = \begin{cases} \frac{2(s+1)}{3s}, & \text{if} \quad 1 \leq s \leq \frac{4}{3} \\ \frac{s+1}{2}, & \text{if} \quad \frac{4}{3} \leq s \leq 2 \\ \frac{s+1}{s} & \text{if} \quad s \geq 2 \end{cases}. \tag{3.12}$$

Furthermore, a similar approach is used for the problem with 3 uniform machines. Given a problem instance of $Q3 \mid \mid C_{\max}$, where there are two faster machines, so that $s_1 = s_2 = s$, $s \geq 1$, and $s_3 = 1$, Soper and Strusevich (2014a) show that for Class 1 instances, where $C_{\max}(S_p^*) = p_1/s$, the power of preemption is bounded by (3.12). All instances of Class 2 will also belong to Class 1, since $\frac{p_1+p_2}{2s} \leq \frac{p_1}{s}$. For Class 3, the power of preemption for the three-machine problem with two fast machines, where

$$C_{\max}\left(S_p^*\right) = \frac{p\left(N\right)}{2s+1},$$

the power preemption is shown to be bounded by $\tilde{\Phi}\left(s\right)$, such that

$$\tilde{\Phi}(s) = \begin{cases} \frac{2s+1}{2s}, & \text{if} \quad 1 \leq s \leq \frac{3}{2} \\ \frac{2s+1}{3}, & \text{if} \quad \frac{3}{2} \leq s \leq 2 \\ \frac{2(2s+1)}{3s} & \text{if} \quad s \geq 2. \end{cases}. \tag{3.13}$$

For the version of the three uniform machine problem with two slower machines, such that $s_1 = s$, $s \geq 1$, and $s_2 = s_3 = 1$, Soper and Strusevich (2014a) show that for Class 1 instances, where $C_{\max}\left(S_p^*\right) = p_1/s$, the power of preemption ρ_3, is bounded by

$$\rho_3 \leq \begin{cases} \frac{4}{3}, & \text{if} \quad 1 \leq s \leq 2 \\ \frac{3s+2}{3s}, & \text{if} \quad s \geq 2, \end{cases}.$$

Moreover, for Class 2 instances of the problem with two slower machines, where

$$C_{\max}\left(S_p^*\right) = \frac{(p_1+p_2)}{(s+1)},$$

it is proven that the power of preemption is bounded by

$$\rho_3 \leq \frac{C_{\max}(S_{np})}{C_{\max}(S_p^*)} \leq \begin{cases} \frac{s+1}{2}, & \text{if} \quad 1 \leq s \leq 2 \\ \frac{s+1}{s}, & \text{if} \quad s \geq 2 \end{cases}.$$

Finally, for Class 3 instances, where the makespan of the preemptive schedule is given by the average machine load, so that

$$C_{\max}\left(S_{np}^{*}\right)=\frac{p\left(N\right)}{s+2},$$

it is proven that for the case where $p_2 \geq \frac{1}{3}p(N)$, the power of preemption is bounded by

$$\rho_3 \leq \begin{cases} \frac{s+1}{2}, & \text{if} \quad 1 \leq s \leq 2 \\ \frac{s+1}{s}, & \text{if} \quad 2 \leq s \leq \frac{1}{2}\left(\sqrt{13}+1\right) = 2.3028 \\ \frac{s+2}{3} & \text{if} \quad \frac{1}{2}\left(\sqrt{13}+1\right) \leq s \leq 3. \\ \frac{s+2}{s} & \text{if} \quad s \geq 3 \end{cases}$$

while for the case where $p_2 \leq \frac{1}{3}p(N)$, the bound is given by

$$\rho_3 \leq \begin{cases} \frac{s+2}{2s}, & \text{if} \quad 1 \leq s \leq \frac{5}{4} \\ \frac{2(s+2)}{5}, & \text{if} \quad \frac{5}{4} \leq s \leq \frac{3}{2} \\ \frac{s+2}{s+1} & \text{if} \quad \frac{3}{2} \leq s \leq 2 \\ \frac{s+2}{3} & \text{if} \quad 2 \leq s \leq 3 \\ \frac{s+2}{s} & \text{if} \quad s \geq 3 \end{cases}.$$

3.2.3 Minimizing Sum of Completion Times

Consider the problem of minimizing the sum of completion times. Given an optimal non-preemptive schedule S_{np}^{*} and an optimal preemptive schedule S_{np}^{*} for this problem, the power of preemption is defined as the supremum of the ratio

$$\frac{\sum C_j\left(S_{np}^{*}\right)}{\sum C_j\left(S_{p}^{*}\right)} \leq \rho_m$$

across all instances. For the general case on m unrelated machines, Epstein et al. (2016) show that the power of preemption for this problem is bounded by $R \simeq 1.39795\ldots$, so that

$$\rho_m \leq R \simeq 1.39795\ldots. \tag{3.14}$$

In order to obtain this bound, Epstein et al. (2016) consider the following: As the value of the power of preemption is defined as the supremum of an infinite set of values, a tight instance may not necessarily exist. In such cases, there is a sequence of instances whose sequence of cost ratios approaches ρ_m. A sequence of this type is called a tight sequence. Let I be an instance of a tight sequence such that each instance in the sequence is a *good input* with equal numbers of unit-length jobs and machines.

Definition 3.3 (Epstein et al. (2016))**.** *An instance I with n unit-length jobs and m machines is called a good input if it satisfies the following conditions:*

(i) $n \geq m$;

(ii) machine M_1 has speed s, $1 < s \leq n$, while the speed of each of the remaining machines $M_2, \ldots, M_m$ is 1;

(iii) in the optimal non-preemptive schedule $S^*_{np}(I)$ found by Algorithm **Q-NonPreempt** in Section 2.2.2, at least one of the unit speed machines remains idle.

Epstein et al. (2016) prove that any instance I can be converted into a good input, without the cost ratio being decreased.

An optimal preemptive schedule $S^*_p(I)$ can be found in polynomial time by applying the preemptive algorithm described in Section 2.2.2 to instance I, for which the optimal value of the total flow time is

$$\sum C_j\left(S^*_p\right) = s\left(\frac{s-1}{s}\right)^{n-1} + n + 1 - s. \tag{3.15}$$

An optimal non-preemptive schedule $S^*_{np}(I)$ is also obtained in polynomial time, in this case by applying the non-preemptive algorithm described in Section 2.2.2. Epstein et al. (2016) show that the resulting optimal value of the objective function is given by

$$\sum C_j\left(S^*_{np}\right) \leq n - \frac{s}{2} + \frac{1}{2}, \tag{3.16}$$

where in the case in which machines have integer speeds, (3.16) holds as an equality.

With respect to (3.15) and (3.16), the following expression the upper bound of the power of preemption is obtained

$$\rho_m \leq \sup_{1 \leq s \leq n} \frac{n - \frac{s}{2} + \frac{1}{2}}{\left(\frac{s-1}{s}\right)^{n+1} + n + 1 - s}. \tag{3.17}$$

The maximum in the right hand side of (3.17) is achieved by a tight sequence where the number of unit-sized jobs, n, goes to infinity. In a tight sequence, a good input instance is defined by a pair (n, s), where n is the number of jobs and machines, while $s \leq n$ is the speed of the faster machine. Consider all pairs (n, s) along a tight sequence.

Since $1 \le s \le n$, it follows that the sequence of values s/n is bounded. In this case, there is a converging subsequence, whose limit is denoted by μ. The instances related to the ratios s/n of the converging subsequence, also form a tight sequence where the number of jobs goes to infinity. Thus

$$\rho_m \le \sup_{0\le\mu\le1} \lim \frac{n - \frac{\mu n}{2} + \frac{1}{2}}{(\mu n)\left(\frac{\mu n - 1}{\mu n}\right)^{n+1} + n + 1 - \mu n} = \sup_{0\le\mu\le1} \frac{1 - \frac{\mu}{2}}{\mu e^{-\frac{1}{\mu}} - \mu + 1}.$$

The numerical value of function $R(\mu)$, where

$$R(\mu) = \frac{1 - \frac{\mu}{2}}{\mu e^{-\frac{1}{\mu}} - \mu + 1},$$

reaches its maximum for $\mu_0 \simeq 0.7959\ldots$, which gives an upper bound $R(\mu_0)$ on the power of preemption. The value of $R(\mu_0)$ is approximately equal to $1.39795\ldots$.

To demonstrate the tightness of (3.14), consider a tight sequence such that instance I_l is associated with a pair of integers (n_l, s_l). Instance I_l is a good input that contains n_l unit-sized jobs and n_l machines, such that the speed of the faster machine is s_l, $1 \le s_l \le n$, while the remaining machines have speed 1. Furthermore

$$\lim_{l\to\infty} s_l = +\infty; \lim_{l\to\infty} n_l = +\infty; \lim_{l\longrightarrow\infty} \frac{s_l}{n_l} = \mu_0.$$

Although bound (3.14) is a global upper bound for the power of preemption for problem $Qm \,|\,|\sum C_j$, Epstein et al. (2016) show that the value of the upper bound can be reduced for a fixed number of machines. Specifically, for the two-uniform machine problem, $Q2 \,|\,|\sum C_j$, they prove that bound

$$\rho_2 \le \frac{6}{5}, \tag{3.18}$$

holds and is tight. Tightness of this bound is demonstrated by considering an instance with two unit-size jobs and $s = 2$. In this case it holds that $C_1\left(S^*_{np}\right) = C_1\left(S^*_p\right) = 1/2$, $C_2\left(S^*_{np}\right) = 1$ and $C_2\left(S^*_p\right) = 3/4$, so that $\sum C_j\left(S^*_{np}\right) = 3/2$ and $\sum C_j\left(S^*_p\right) = 5/4$, leading to the cost ratio of $\sum C_j\left(S^*_{np}\right) / \sum C_j\left(S^*_p\right) = 6/5$.

3.3 Power of Preemption on Unrelated Machines

3.3.1 Minimizing Makespan

For the problem of minimizing the makespan on unrelated machines described in Section 2.3.2, we showed that an optimal preemptive schedule S^*_{np} is found in polynomial time by the two-stage procedure due to Lawler and Labetoulle (1987), while the problem of finding a non-preemptive schedule S_p is NP-hard, and the best approximation results are due to the rounding procedure by Shmoys and Tardos (1993). For this problem Correa, Skutella and Verschae (2012) give the power of preemption as

$$\rho_m = \frac{C_{\max}\left(S^*_{np}\right)}{C_{\max}\left(S^*_{p}\right)} \leq 4. \tag{3.19}$$

They further show that the above bound on the power of preemption holds by describing a procedure for constructing problem instances, for which the above bound holds as an equality. This procedure is described next. Given some instance I of the makespan problem on unrelated machines, the linear programming formulation (2.20) results in a set of values t_{ij}, which as discussed in Section 2.3.2 define the processing requirement of every job J_j on machine M_i. The LP also results in a value C which gives the optimal value of the makespan of a preemptive schedule. Based on this linear programming formulation by Lawler and Labetoulle, Lin and Vitter (1992) show that the rounding technique of Shmoys and Tardos (1993) for the assignment problem can be applied to this set of results, in order to obtain a non-preemptive schedule S_{np} with makespan at most $4C$. This procedure is described next.

Consider an optimal solution x_{ij}, C to (2.20), where x_{ij} defines the portion of job J_j assigned to machine M_i, so that $x_{ij} = t_{ij}/p_{ij}$, and C the makespan of an optimal preemptive schedule. For some fixed value $\beta > 1$, a modified solution x'_{ij}, βC is then defined, so that

$$x'_{ij} = \left\{ \begin{array}{ll} x'_{ij} = 0 & \text{for} \quad p_{ij} > \beta C \\ x'_{ij} = \frac{x_{ij}}{X_j} & \text{for} \quad p_{ij} \leq \beta C \end{array} \right\},$$

where X_j denotes the total portion of J_j which is assigned to machines, for which the processing time of J_j is at most βC. Consider a set of r machines $\mathcal{M}_{r(j)}$, so that the processing time of a job J_j is at most βC for any machine $M_{i(u)} \in \mathcal{M}_{r(j)}$, $1 \leq u \leq r$. Let $x_{i(u)j}$ denote the portion of J_j assigned to machine $M_{i(u)} \in \mathcal{M}_{r(j)}$, thus X_j is given by

$$X_j = \sum_{u=1}^{r} x_{i(u)j},\ 1 \leq j \leq n,\ 1 \leq i \leq m.$$

Rounding the modified solution set x'_{ij} with the procedure by Shmoys and Tardos (1993), yields a set of 0-1 integer values $\hat{x}_{ij} \in \{0, 1\}$, which are a feasible solution to (2.20), and thus a feasible solution to $Rm \,|\,|\, C_{\max}$, where

$$\sum_{j=1}^{n} \hat{x}_{ij} p_{ij} \leq \sum_{j=1}^{n} x'_{ij} p_{ij} + \max \left\{ p_{ij} \middle| \, x'_{ij} > 0 \right\} \leq \frac{\beta^2}{\beta - 1} C.$$

Observe that the minimum for $\beta^2 / (\beta - 1)\, C$ is achieved for $\beta = 2$, thus the makespan of the rounded solution is at most $4C$.

In order to demonstrate the tightness of this bound, Correa, Skutella and Verschae (2012) further describe a procedure of constructing an instance $I(\beta, \varepsilon)$, $2 \leq \beta < 4$ and $\varepsilon > 0$ such that $1/\varepsilon$ is a positive integer, for which bound (3.19) is tight. For an instance $I(\beta, \varepsilon)$ the makespan of the preemptive schedule is at most $(1 + \varepsilon)\, C$, and that of a non-preemptive schedule is at least βC. This procedure, new instances are produced iteratively, until the desired instance is found. In every iteration the makespan of the non-preemptive schedule is increased, while the makespan of the preemptive schedule is kept equal to $(1 + \varepsilon)\, C$.

Let I_ζ, $\zeta \geq 0$, be an instance obtained at the ζ-th iteration of the procedure by Correa, Skutella and Verschae (2012). Each instance $I_\zeta = (N_\zeta, \mathcal{M}_\zeta)$ is defined by a set of n_ζ jobs $N_\zeta := \left\{ J_{j(\zeta,1)}, J_{j(\zeta,2)}, \ldots, J_{j(\zeta, n_\zeta)} \right\}$ and a set of m_ζ machines $\mathcal{M}_\zeta := \left\{ M_{i(\zeta,1)}, M_{i(\zeta,2)}, \ldots, M_{i(\zeta,3)} \right\}$.

Algorithm Generate Instance (Correa, Skutella and Verschae (2012))

Step 1. Construct I_0 and f_0, and let $n = 0$.

Step 2. Construct and instance Y_{n+1} consisting of $1/\varepsilon$ copies of instance I_n, that are denoted as I_n^l, for $l = 1, 2, \ldots, 1/\varepsilon$, where the copy of machine $M_{i(n+1)}$ belonging to I_n^l is denoted by $M_{i(n+1;l)}$.

Step 3. Create $1/\varepsilon$ copies of Y_{n+1}, Y_{n+1}^k, $k = 1, 2, \ldots, 1/\varepsilon$. Denote the l-th copy of instance I_n belonging to instance Y_{n+1}^k as I_n^{lk}, and the copy of machine $M_{i(n+1)}$ that belongs to instance I_n^{lk} as $M_{i(n+1;l;k)}$.

Step 4. Create $1/\varepsilon$ new jobs, $J_{j(n+1;k)}$, $k = 1, 2, \ldots, 1/\varepsilon$, and let $p_{i(n+1;l;k)j(n+1;k)} = C\,(\beta - 1/f_n)$ for all $k, l = 1, 2, \ldots, 1/\varepsilon$, and $+\infty$ for all other machines.

The assignment variables for the new jobs are defined as

$$x_{i(n+1;l;k)j(n+1;k)} = \frac{\varepsilon}{\beta - 1/f_n}, \text{ where } k, l = 1, 2, \dots, 1/\varepsilon.$$

Thus the unassigned fraction of each job $j\,(n+1;k)$ equals

$$\begin{aligned} f_{n+1} &= 1 - \sum_{l=1}^{1/\varepsilon} x_{i(n+1;l;k)j(n+1;k)} \\ &= \frac{(\beta-1)\,f_n - 1}{\beta f_n - 1}. \end{aligned}$$

Step 5. Create a new machine $M_{i(n+2)}$, and define $p_{i(n+2)j(n+1;k)} = \varepsilon C/f_{n+1}$ for all $k = 1, 2, \dots, 1/\varepsilon$, and $+\infty$ for all other machines.

Step 6. Call the instance constructed so far I_{n+1}, and increment n.

Step 7. If $f_n > 1/(\beta - 1)$, repeat Step 2. Otherwise go to Step 8.

Step 8. Make $1/\varepsilon$ copies of I_n, I_n^l for $l = 1, 2, \dots, 1/\varepsilon$, and call $M_{i(n+1;l)}$ the copy of machine $M_{i(n+1)}$ of instance I_n^l.

Step 9. Create a new job $j\,(n+1)$, and define $p_{i(n+1;l)j(n+1)} := C\,(\beta - 1/f_n)$ and $x_{i(n+1;l)j(n+1)} = \varepsilon$.

Step 10. Return instance I_{n+1}.

3.3.2 Minimizing Sum of Completion Times

Consider the problem of minimizing the total completion time on unrelated machines. Let $\sum C_j\left(S_p^*\right)$ denote the value of the objective function of an optimal preemptive schedule, and $\sum C_j\left(S_{np}^*\right)$ the value of the optimal non-preemptive schedule. As discussed in Section 2.3.3, this problem has the characteristic property where the non-preemptive version, i.e. problem $Rm\,|\,|\sum C_j$ is solvable in polynomial time, while the preemptive version is both NP-hard and APX-hard. For the weighted version of this problem denoted by $Rm\,|\,|\sum w_j C_j$,

Sitters (2008) presents an algorithm that produces a non-preemptive schedule for which the sum of weighted completion times is no more than 1.81 times the value of the optimal preemptive schedule, thus the bound for power of preemption ρ_m is given by

$$\rho_m = \frac{\sum w_j C_j\left(S_{np}^*\right)}{\sum w_j C_j\left(S_p^*\right)} < 1.81.$$

For a given preemptive schedule S, the *density function* of job J_j defines the speed at which the job is being processed at time t, and is denoted by $f_j(t)$. The *mean busy time,* B_j, is defined as the average time at which job J_j is being processed, so that

$$B_j = \int_0^T f_j(t)\, t dt,$$

where T is any upper bound on the completion time of J_j. Furthermore, the *process time* P_j of J_j is defined as the total amount of time that the job is being processed on all machines. Sitters (2008) proves that for any instance I of $R\,|pmtn|\sum w_j C_j$, the inequality

$$\min\left(\sum_{j=1}^{n} w_j P_j + w_j B_j\right) < 1.81 \min\left(\sum_{j=1}^{n} w_j C_j\right). \tag{3.20}$$

holds. Based on this result, Sitters (2008) initially provides a convex quadratic program for which the optimal value is a lower bound on the left hand side of (3.20), and shows that any solution Z of the program can be rounded with the application of randomized rounding, to obtain a feasible non-preemptive schedule for which the expected sum of weighted completion times is at most Z.

CHAPTER 4

Power of Limited Preemption

For scheduling problems with a limited number of preemptions, we denote by θ the maximum number of preemptions allowed. Moreover, given a θ-preemptive schedule is defined as a schedule with at most θ preemptions, and is denoted by $S_{(\theta)}$. Given a scheduling problem, the power of limited preemption is defined as the maximum ratio $\Phi\left(S^*_{(\theta)}\right)/\Phi\left(S^*_p\right)$ across all instances of the problem. For a scheduling problem with m machines, we denote the power of limited preemption by $\rho_m^{(\theta)}$.

$$\frac{\Phi\left(S^*_{(\theta)}\right)}{\Phi\left(S^*_p\right)} \leq \rho_m^{(\theta)} \tag{4.1}$$

4.1 Identical Machines

In this section, we review the results on limited preemptions on identical parallel machines.

4.1.1 Minimizing Makespan

Consider the problem of minimizing the makespan on identical parallel machines for the case where only one preemption is allowed. It is easy to see that the two-machine version of this problem, denoted by $P2\left|\#pmtn \leq 1\right| C_{\max}$, can be solved in polynomial time. This is due to the fact that, as shown by McNaughton (1959), an optimal preemptive schedule on identical machines requires at most $m-1$ preemptions, thus a single-preemptive schedule for problem for the case $m = 2$ is bound to be an optimal preemptive schedule.

For three identical machines, the single-preemptive makespan problem is NP-hard. We prove this statement in the following lemma.

Lemma 4.1. *Problem $P3\,|\#pmnt \leq 1|\,C_{\max}$ is NP-hard in the ordinary sense.*

Proof: We provide reduction from the following NP-complete problem.

SUBSET-SUM: Given r positive integers e_j and an integer E such that $e_j \leq E$ and $\sum_{j=1}^{r} e_j = 3E$, does there exist a subset R' of the index set $R = \{1, 2, \ldots, r\}$ such that $e(R') = E$?

This problem is version of the subset-sum problem by Garey and Johnson (1979).

Given an instance of SUBSET-SUM, define the following instance of a decision version of problem $P3\,|\#pmnt = 1|\,C_{\max}$ with the set $N = \{1, 2, \ldots, n\}$:

$$n = r,\ p_j = e_j, j \in N.$$

It is required to verify whether there exists a schedule $S_{(1)}$ with a single preemption with $C_{\max}\left(S_{(1)}\right) \leq E$.

Suppose that SUBSET-SUM has a solution and R' is the required subset. Then assign the jobs $j \in R'$ to machine M_1, and schedule the remaining jobs on M_2 and M_3 by McNaughton's algorithm. In the resulting schedule there is at most one preemption (one job between M_2 and M_3), and each machine completes exactly at time E.

Suppose now that schedule $S_{(1)}$ exists. Since $p\,(N) = 3E$, it follows that $C_{\max}\left(S_{(1)}\right) = E$, and each machine is permanently busy processing its job in the time interval $[0, E]$. Since there is only one preemption in $S_{(1)}$, there is a machine on which the jobs are processed without preemption. Without loss of generality assume that such a machine is machine M_3. Then denoting the set of jobs processed on M_3 in schedule $S_{(1)}$ by R', we obtain a solution to SUBSET-SUM. □

This result can be extended to an arbitrary number of machines. Braun and Schmidt (2003) claim that for an arbitrary number of machines, problem $Pm\,|\#pmnt \leq m-2|\,C_{\max}$ is NP-hard in the strong sense.

Now, we pass to the problem of minimizing the makespan on m identical machines with a limited number of preemptions.

An optimal preemptive schedule for this problem is obtained in $O\,(n)$ time with application of McNaughton's wrap-around algorithm, requiring at most $m-1$ preemptions. Braun and Schmidt (2003) prove that the bound

$$\rho_m^{(\theta)} \leq 2\frac{m}{m+\theta+1} \tag{4.2}$$

on the power of limited preemption holds and is tight.

In order to demonstrate the tightness of the above bound, consider an instance I consisting of m identical machines and n identical jobs, so that

$$n = m + \theta + 1, \tag{4.3}$$

and

$$p_1 = p_2 = \ldots = p_n = p. \tag{4.4}$$

From (4.3) and (4.4), the total processing time of the instance I is given by

$$p\left(N\right) = p \cdot \left(m + \theta + 1\right),$$

i.e., for instance I the average machine load given by (2.5) exceeds the processing time of any individual job in N. Thus, it follows from Definition 2.1 that I belongs to Class 1 of instances, and

$$C_{\max}\left(S_p^*\left(I\right)\right) = \frac{p\left(m + \theta + 1\right)}{m}.$$

Braun and Schmidt (2003) prove that in an optimal θ-preemptive schedule for instance I, the terminating job J_k is not preempted. Furthermore, the machine which processes job J_k, may or may not allow preemption. In either case, one of the machines in the schedule is bound to process two jobs, thus for this instance the optimal θ-preemptive schedule $S_{(\theta)}^*\left(I\right)$, has makespan

$$C_{\max}\left(S_{(\theta)}^*\left(I\right)\right) = 2p.$$

From the above, it is determined that the power of limited preemption for instance I is given by (4.2).

An algorithm for obtaining θ-preemptive schedules for which (4.4) holds is described by Jiang et al. (2014). Their algorithm uses a hybrid approach which utilizes List Scheduling and McNaughton's wrap around rule. For some instance I, the algorithm splits at most θ jobs into two fraction. Each fraction is then scheduled on one of the θ first machines which allow preemption, while the remaining jobs are scheduled following the list scheduling algorithm on the remaining $m - \theta$ machines on which preemption is not permitted.

Algorithm JH-Pm-θ (Jiang et al. (2014))

INPUT: An instance I for problem $Pm\,|\#pmtn \leq \theta|\,C_{\max}$

OUTPUT: A θ-preemptive schedule $S_{(\theta)}$ which satisfies (4.2)

Step 1. Compute $C_{\max}(S^*)$ according to (2.7).

Step 2. Determine job J_δ such that

$$\sum_{j=1}^{\delta} p_j \geq \theta R_\theta C_{\max}(S^*),$$

and

$$\sum_{j=1}^{\delta-1} p_j < \theta R_\theta C_{\max}(S^*),$$

where

$$R_\theta = \frac{2m}{m+\theta+1}.$$

Then split J_δ into J'_δ and J''_δ, with processing times p'_δ and p''_δ respectively, so that

$$p'_k = \theta R_\theta C_{\max}(S^*) - \sum_{j=1}^{k} p_j.$$

Step 3. Schedule jobs $J_1, J_2, \ldots, J_{\delta-1}, J'_k$, preemptively on the first θ machines using McNaughton's wrap around.

Step 4. Schedule jobs $J''_k, J_{k+1}, \ldots, J_n$, by initially assigning each of the $m-\theta$ largest jobs on one of the last $m-\theta$ machines. Assign the remaining jobs non-preemptively on the $m-\theta$ machines using List Scheduling.

Step 5. Schedule job J''_δ so that it is the first job to be processed on the machine on which it has been assigned.

4.2 Uniform Machines

In this section we present our results on a power of a single preemption on uniform machines for the problem of minimizing the makespan.

4.2.1 Complexity

For two uniform machines, we consider the problem of finding a schedule with a single preemption that minimizes the makespan. Specifically, we show that problem

$Q2\left|\#pmnt \le 1\right| C_{\max}$ can be solved in polynomial time. In this problem there are two uniform machines M_1 and M_2, with speeds s_1 and s_2 respectively, such that $s_1 = s$, $s \ge 1$, and $s_2 = 1$. Furthermore, jobs are numbered in LPT order according to (2.1). As discussed in Section 2.2.1, if there no restriction on the number of preemptions, problem $Q2\left|pmnt\right| C_{\max}$ is solvable in $O(n)$ time, and the makespan of the optimal preemptive schedule is given by (2.16).

Consider the following algorithm for obtaining a schedule $S^*_{(1)}$ with a single preemption for the problem on two uniform machines.

Algorithm Q2(1)

INPUT: An instance I for problem $Q2\left|\#pmtn \le 1\right| C_{\max}$

OUTPUT: An optimal1-preemptive schedule $S_{(1)}(I)$

Step 1. Compute T_2 according to (3.10). If

$$p_1 < sT_2,$$

go to Step 2. Otherwise, assign J_1 to M_1 and the remaining $n-1$ jobs to M_2 and go to Step 5.

Step 2. Scanning the jobs in the order of their numbering, find job k such that

$$\sum_{j=1}^{k-1} p_j < sT_2, \ \sum_{j=1}^{k} p_j \ge sT_2.$$

Step 3. Compute

$$x_k = sT_2 - \sum_{j=1}^{k-1} p_j, \ y_k = p_k - x_k.$$

If

$$y_k > \frac{1}{s}\sum_{j=1}^{k-1} p_j,$$

go to Step 4; otherwise output the following schedule $S_{(1)}$: on M_1 the jobs $1, \dots, k-1$ are processed in any order, followed by a part of job k for x_k/s time units; on M_2 a part of job k for y_k time units, followed by an arbitrary sequence of jobs $k+1, \dots, n$, and go to Step 5.

Step 4. Compute

$$y'_k = \frac{1}{s}\sum_{j=1}^{k-1} p_j, \ x'_k = p_k - y'_k.$$

Output the following schedule $S_{(1)}$: on M_1 the jobs $1, \ldots, k-1$ are processed in any order, followed by a part of job k for x'_k/s time units; on M_2 a part of job k for y'_k time units, followed by an arbitrary sequence of jobs $k+1, \ldots, n$.

Step 5. Stop.

It is simple to see that Algorithm Q2(1) requires $O(n)$ time. In order to show that this problem is polynomially solvable, it is sufficient to prove that a solution produced by the algorithm is optimal for all problem instances. This is proven in the following theorem.

Theorem 4.1. *Schedule $S_{(1)}$ found by Algorithm Q2Pr1 is optimal for problem $Q2\,|\#pmnt \leq 1|\, C_{\max}$.*

Proof: If $p_1 \geq sT_2$ then

$$\frac{p_1}{s} \geq \frac{p(N)}{s+1},$$

so that

$$p_1(s+1) \geq sp(N),$$

i.e.,

$$\frac{p_1}{s} \geq p(N \backslash \{J_1\})$$

For schedule $S_{(1)}$ found in Step 1, machine M_1 terminates at time p_1/s, while machine M_2 terminates at time $p(N\backslash\{J_1\})$. Thus,

$$C_{\max}\left(S_{(1)}\right) = p_1/s,$$

which implies that schedule $S_{(1)}$ is a non-preemptive, optimal schedule for problem $Q2\,|pmnt|\, C_{\max}$, and therefore for problem $Q2\,|\#pmnt \leq 1|\, C_{\max}$.

We come to Step 2 if

$$p_1 < sT_2.$$

Since

$$\frac{1}{s}\sum_{j=1}^{n} p_j > \frac{1}{s+1}\sum_{j=1}^{n} p_j = T_2,$$

it follows that job J_k exists.

For schedule $S_{(1)}$ found in Step 3, both machines terminate at time T_2. This schedule is feasible since the part of the preempted job J_k completes on M_2 at time

$$y_k \leq \frac{1}{s}\sum_{j=1}^{k-1} p_j,$$

while this job starts on M_1 at time $\frac{1}{s}\sum_{j=1}^{k-1} p_j$, which produces no clash. Since $C_{\max}\left(S_{(1)}\right) = T_2$, schedule $S_{(1)}$ is a non-preemptive optimal schedule for problem $Q2\left|pmnt\right|C_{\max}$, and therefore for problem $Q2\left|\#pmnt \leq 1\right|C_{\max}$.

We come to Step 4 if

$$y_k > \frac{1}{s}\sum_{j=1}^{k-1} p_j,$$

which implies that

$$y_k + \frac{x_k}{s} > T_2.$$

Since

$$p_k = y_k + x_k > y_k + \frac{x_k}{s},$$

we deduce from the LPT numbering of jobs that

$$p_1 \geq \cdots \geq p_{k-1} \geq p_k > T_2.$$

Let $S'_{(1)}$ denote the best schedule for problem $Q2\left|\#pmnt \leq 1\right|C_{\max}$, in which the jobs of set $N_k = \{J_1, \cdots, J_k\}$ are processed without preemption. If all these jobs are assigned to machine M_1 then $C_{\max}\left(S'_{(1)}\right) = \frac{1}{s}p\left(N_k\right) > T_2$. If at least one of these jobs is assigned to be processed on machine M_2 then that job must be the shortest job J_k, so that $C_{\max}\left(S'_{(1)}\right) = p_k > T_2$. However, none of these schedules is optimal for problem $Q2\left|\#pmnt \leq 1\right|C_{\max}$, since a better schedule can be obtained by preemptively processing one of the jobs of set N_k.

Take a job $J_\ell \in N_k$. In order to produce a schedule that is better than $S'_{(1)}$, we introduce a schedule $S^{(\ell)}$ which is the best schedule in which job J_ℓ is processed preemptively. In schedule $S^{(\ell)}$ the jobs of set $N_k \backslash \{J_\ell\}$ are processed on M_1without preemption, and job J_ℓ is preempted to be processed without clashes on M_2 for y'_ℓ time units and on M_1 for x'_ℓ/s time units, where

$$y'_\ell = \frac{1}{s}p\left(N_k \backslash \{J_\ell\}\right),\ x'_\ell = p_\ell - y'_\ell.$$

In schedule $S^{(\ell)}$machine M_1 terminates at time

$$\begin{aligned}
C_{\max}\left(S^{(\ell)}\right) &= y'_\ell + x'_\ell/s = y'_\ell + \left(p_\ell - y'_\ell\right)/s \\
&= \frac{p_\ell}{s} + y'_\ell\left(1-\frac{1}{s}\right) = \frac{p_\ell}{s} + \frac{p\left(N_k\right)-p_\ell}{s}\left(1-\frac{1}{s}\right) \\
&= \frac{p_\ell + (s-1)\,p\left(N_k\right)}{s^2}
\end{aligned}$$

The right-hand side of the above expression increases in p_ℓ, thus it takes it minimum for $\ell = k$.

Clearly, for $\ell = k$ we have that $C_{\max}\left(S^{(\ell)}\right) = y'_k + x'_k/s < p_k$ and

$$C_{\max}\left(S^{(\ell)}\right) = \frac{1}{s}\left(\sum_{j=1}^{k-1} p_j + x'_k\right) < \frac{1}{s}\sum_{j=1}^{k} p_j,$$

so that $C_{\max}\left(S^{(\ell)}\right) < C_{\max}\left(S'_{(1)}\right)$. Since schedule $S_{(1)}$ found in Step 4 is actually schedule $S^{(\ell)}$ for $\ell = k$, we deduce that $S_{(1)}$ is optimal for $Q2\left|\#pmnt \leq 1\right| C_{\max}$.

We note that the optimal makespan can also be written as

$$C_{\max}\left(S^{(k)}\right) = \frac{\left(\frac{s-1}{s}\right)\sum_{i=1}^{k-1} p_j + p_k}{s}. \tag{4.5}$$

□

For the problem with an arbitrary number of machines, where $m > 2$, it follows from the equivalent problem on m identical machines that problem $Qm\left|\#pmtn \leq 1\right| C_{\max}$ is NP-hard.

4.2.2 Power of single preemption on two uniform machines

For problem $Q2\left|\#pmtn \leq 1\right| C_{\max}$, Jiang et al. (2014) show that the power of limited preemption for problems where a single preemption is allowed, is bounded by

$$\rho_2^{(1)} \leq \frac{2s^2 + s - 1}{2s^2}. \tag{4.6}$$

Consider an instance I of this problem with two unit length jobs $J_1,\ J_2$, such that $p_1 = p_2 = 1$. In this case, it can be simply observed that a feasible schedule $S^*_{(1)}(I)$ is obtained by scheduling J_1 entirely on the fastest machine M_1, while the processing time of J_2 is split into two fractions $p_2' = 1/s$ and $p_2'' = 1 - 1/s$, where p_2' is assigned to machine M_1 and p_2'' on machine M_2, so that $C_{\max}\left(S^*_{(1)}(I)\right)$ satisfies (4.6).

The following algorithm by Jiang et al. (2014) finds a schedule $S^*_{(1)}(I)$ for which bound (4.6) holds.

Algorithm JH-Q2(1) (Jiang et al. (2014))

INPUT: An instance I for problem $Q2\,|\#pmtn \le 1|\,C_{\max}$

OUTPUT: A 1-preemptive schedule $S_{1p}(I)$ which satisfies (4.6)

Step 1. Set J_1 as the largest job in N, so that $p_1 = \max\{p_j|\ 1 \le j \le n\}$. Compute the optimal makespan $C_{\max}(S^*(I))$, according to (2.16)

Step 2. Find job J_k, such that $k = \min\left\{j|\sum_{i=1}^{j} p_i > s \cdot R_1 \cdot C_{\max}(S^*(I))\right\}$, where

$$R_1 = \frac{2s^2 + s - 1}{2s^2}.$$

Split J_k into J_k' and J_k'' so that

$$p_k' = s \cdot R_1 \cdot C_{\max}(S^*(I)) - \sum_{j=1}^{k-1} p_j,$$

and

$$p_k'' = p_k - p_k'.$$

Step 3. Assign jobs $J_1, J_2, \ldots, J_{k-1}, J_k'$ on machine M_1, and jobs $J_k'', J_{k+1}, \ldots, J_n$, on machine M_2.

Step 4. Stop.

This algorithm runs in $O(n)$ time.

4.2.3 Power of single preemption on m uniform machines

In this section we determine the general bounds for the power of a single preemption for the problem on m uniform machines.

Consider an instance $I = (\mathcal{L}_n, \mathcal{M}_m)$ of the makespan problem on m uniform machines, as defined in Section (2.2). A 1-preemptive schedule $S_{(1)}(I)$ for this problem is defined by

(i) a job $J_\ell \in N$ which is processed with preemption on two machines $M_{\ell'}$ and $M_{\ell''}$ such that $1 \leq \ell' < \ell'' \leq m$; the actual processing times of job J_ℓ on these machines are equal to $x_\ell / s_{\ell'}$ and $y_\ell / s_{\ell''}$, where $x_\ell + y_\ell = p_\ell$;

(ii) a partition of set $N \backslash \{J_\ell\}$ into m subsets $N_1, N_2, \ldots, N_m$, where the jobs of set N_i are assigned to be processed on machine M_i, $1 \leq i \leq m$.

Notice that even in an optimal schedule some of the subsets N_i can be empty, since it may be counterproductive to assign jobs to very slow machines. It is possible that for a particular instance it is optimal not to preempt any job, in which case a schedule $S_{(1)}(I)$ is defined by a partition of set N into m subsets $N_1, N_2, \ldots, N_m$. If there is a preempted job J_ℓ in schedule $S_{(1)}(I)$ then the jobs assigned to machines $M_{\ell'}$ and $M_{\ell''}$ must be arranged in a such a way that job J_ℓ is not processed by more than one machine at a time

Next, we show that for problem $Qm\,|\#pmtn \leq 1|\,C_{\max}$, the power of limited preemption is bounded by

$$\rho_m^{(1)} \leq 2 - \frac{2}{m}. \tag{4.7}$$

The following algorithm finds a single-preemptive schedule $S_{(1)}$, such that the bound

$$\frac{C_{\max}\left(S_{(1)}\right)}{C_{\max}\left(S_p^*\right)} \leq \rho_m^{(1)}$$

holds, and is tight.

Algorithm Qm(1)LPT

INPUT: An instance I for problem $Qm\,|\#pmtn \leq 1|\,C_{\max}$

OUTPUT: A 1-preemptive schedule $S_{(1)}(I)$ which satisfies (4.7)

Step 1. Compute $C_{\max}\left(S_p^*\right) = \max\{T_u | 1 \leq u \leq m\}$ and the processing time bound $B = \rho_m^{(1)} C_{\max}\left(S_p^*\right)$. If required, renumber the jobs in accordance with the LPT rule (2.1) and form the list $\mathcal{L}_n = (p_1, \ldots, p_n)$. Define

$$N_i := \varnothing,\ p(N_i) := 0,\ 1 \leq i \leq m.$$

Step 2. For j from 1 to n do

(a) Take job J_j, the first job in the current list $\mathcal{L}_n$. If there exists a machine M_k such that $p(N_k) + p_j \leq s_k B$, then go to Step 2(b). If $p(N_i) + p_j > s_i B$ for all i, $1 \leq i \leq m$, then if the preemption has not been used go to Step 2(c); otherwise, go to Step 2(d).

(b) Update

$$N_k := N_k \cup \{Jj\}, \; p(N_k) := p(N_k) + p_j$$

and go to Step 2(e).

(c) First, try to process job J_j as the preempted job J_ℓ. If two machines $M_{\ell'}$ and $M_{\ell''}$ such that $1 \leq \ell' < \ell'' \leq m$ can be selected which satisfy the inequalities

$$y_\ell + p(N_{\ell''}) \leq s_{\ell''}B, \tag{4.8}$$

$$\frac{x_\ell}{s_{\ell'}} + \frac{y_\ell}{s_{\ell''}} \leq B, \tag{4.9}$$

where

$$x_\ell = s_{\ell'}B - p(N_{\ell'}),$$

$$y_\ell = p_\ell - x_\ell;$$

then assign job J_ℓ to be processed on machine $M_{\ell'}$ for $x_\ell/s_{\ell'}$ time units and on machine $M_{\ell''}$ for $y_\ell/s_{\ell''}$ time units. Define

$$p(N_{\ell'}) := p(N_{\ell'}) + x_\ell, \; p(N_{\ell''}) := p(N_{\ell''}) + y_\ell.$$

If the required pair of machines cannot be found, then go to Step 2(d).

(d) Find machine M_k such that

$$s_kB - p(N_k) = \max\{s_iB - p(N_i) \,|\, 1 \leq i \leq m\} \tag{4.10}$$

and return to Step 2(b).

(e) Remove job J_j from list $\mathcal{L}_n$.

Step 3. In the found schedule $S_{(1)}(I)$ machine M_i processes the jobs of set N_i, $1 \leq i \leq m$, in an arbitrary order starting from time 0. Additionally, if in the loop in Step 2, Step 2(c) has occurred, then

(i) assign job J_ℓ to be processed on machine $M_{\ell''}$ starting from time 0,

(ii) start the jobs of set $N_{\ell''}$ at time $y_\ell/s_{\ell''}$, and

(iii) assign job J_ℓ to be processed on machine $M_{\ell'}$ starting from time $p(N_{\ell'})/s_{\ell'}$.

Algorithm LPT1 scans the jobs in LPT order and tries to assign each job to the machine where it can be completed by time B. The first time that such an assignment is not possible, the corresponding job is assigned to be processed with preemption (see Step 2(c)). The remaining jobs, if any, are either assigned to be completed by time B wherever possible, or to a specific machine otherwise.

Theorem 4.2. *Given an arbitrary instance $I = (\mathcal{L}_n, \mathcal{M}_m)$, let $S_{(1)}$ be a schedule created by Algorithm Qm(1)LPT using at most one preemption. Then*

$$\frac{C_{\max}\left(S_{(1)}\left(I\right)\right)}{C_{\max}\left(S_p^*\left(I\right)\right)} \leq 2 - \frac{2}{m}. \tag{4.11}$$

Proof: The proof is based on the minimal counterexample technique, often used in worst-case analysis of approximation algorithms. Suppose that the theorem is not true, i.e., there exists an instance $I = (\mathcal{L}_n, \mathcal{M}_m)$, which we call the minimal counterexample, such that

$$\frac{C_{\max}\left(S_{(1)}\left(\mathcal{L}_n, \mathcal{M}_m\right)\right)}{C_{\max}\left(S_p^*\left(\mathcal{L}_n, \mathcal{M}_m\right)\right)} > 2 - \frac{2}{m} \tag{4.12}$$

and no job or machine can be removed from the instance without violating the inequality (4.12).

First consider instances in which the number of jobs is smaller than the number of machines i.e. instances for which the lists $(\mathcal{L}_n, \mathcal{M}_m)$ have $n < m$. Let $\mathcal{M}_n$ be the list of machine speeds obtained from list $\mathcal{M}_m$ by a removal of the $m - n$ slowest machines.

For an instance I, let $S_{(1)}^*(I)$ be an optimal schedule for problem $Q\left|\#pmtn \leq 1\right|C_{\max}$. It is clear that in each schedule $S_{(1)}^*(\mathcal{L}_n, \mathcal{M}_m)$ and $S_p^*(\mathcal{L}_n, \mathcal{M}_m)$, jobs are assigned to at most n fastest machines. Thus,

$$S_{(1)}^*\left(\mathcal{L}_n, \mathcal{M}_m\right) = S_{(1)}^*\left(\mathcal{L}_n, \mathcal{M}_n\right)$$

and

$$C_{\max}\left(S_{(1)}^*\left(\mathcal{L}_n, \mathcal{M}_m\right)\right) = C_{\max}\left(S_{(1)}^*\left(\mathcal{L}_n, \mathcal{M}_n\right)\right),$$

while in the preemptive case,

$$C_{\max}\left(S_p^*\left(\mathcal{L}_n, \mathcal{M}_m\right)\right) = \max\left\{T_u | 1 \leq u \leq n < m\right\} = C_{\max}\left(S_p^*\left(\mathcal{L}_n, \mathcal{M}_n\right)\right).$$

Since for an instance $(\mathcal{L}_n, \mathcal{M}_m)$ with $n < m$ the removal of the $m - n$ slowest machines does not change the value of the power of preemption, the minimal counterexample cannot be one of these instances. Hence, in our search for the minimal counterexample we only need to consider instances in which there are at least as many jobs as machines.

Suppose that in schedule $S_{(1)}(\mathcal{L}_n, \mathcal{M}_m)$ job J_h is the terminal job and machine M_k the critical machine. If $h < n$ then Algorithm Qm(1)LPT assigns some jobs J_j with $j > h$ after job J_h and they complete (on machines other than M_k) earlier than job J_h. Suppose that these jobs are removed from the instance, so that $\mathcal{L}_h = (p_1, p_2, \ldots, p_h)$ is the corresponding list of the processing times. For the modified instance $(\mathcal{L}_h, \mathcal{M}_m)$, we have

$$C_{\max}\left(S_{(1)}(\mathcal{L}_h, \mathcal{M}_m)\right) = C_{\max}\left(S_{(1)}(\mathcal{L}_n, \mathcal{M}_m)\right);\ C_{\max}\left(S_p^*(\mathcal{L}_h, \mathcal{M}_m)\right) \le C_{\max}\left(S_p^*(\mathcal{L}_n, \mathcal{M}_m)\right),$$

so that

$$\frac{C_{\max}\left(S_{(1)}(\mathcal{L}_h, \mathcal{M}_m)\right)}{C_{\max}\left(S_p^*(\mathcal{L}_h, \mathcal{M}_m)\right)} \ge \frac{C_{\max}\left(S_{(1)}(\mathcal{L}_n, \mathcal{M}_m)\right)}{C_{\max}\left(S_p^*(\mathcal{L}_n, \mathcal{M}_m)\right)} > 2 - \frac{2}{m}.$$

Thus, if $h < n$ we deduce that instance $(\mathcal{L}_n, \mathcal{M}_m)$ cannot be the minimal counterexample, and we must have that $h = n$. In other words, for the minimal counterexample $(\mathcal{L}_n, \mathcal{M}_m)$ Algorithm Qm(1)LPT finds a schedule $S_{(1)}(\mathcal{L}_n, \mathcal{M}_m)$ that is terminated by the shortest job J_n. Clearly,

$$p_n \le \frac{1}{n} p(N) \le \frac{1}{m} p(N). \tag{4.13}$$

Suppose that the terminal job J_n is completed on machine M_u, $1 \le u \le m$, and that its completion time exceeds the bound of B, i.e., $p(N_u) > s_u B$.

In schedule $S_{(1)}$, for each machine, find the value G_i such that

$$B = \frac{p(N_i) + G_i}{s_i},\ 1 \le i \le m; i \ne u. \tag{4.14}$$

Let us call the value G_i the gap on machine M_i. We can interpret the gap on a machine as the amount of processing that could be additionally assigned to that machine so that the machine completes at exactly time B. Define, $G_u = 0$, so that there is no gap on the critical machine M_u.

Summing up the equalities (4.14) and the inequality $p(N_u) + G_u > s_u B$, we deduce

$$\begin{aligned}\sum_{i=1}^{m} p(N_i) + \sum_{i=1}^{m} G_i &= p(N) + \sum_{i=1}^{m} G_i > B \sum_{i=1}^{m} s_i \\ &= \left(2 - \frac{2}{m}\right) C_{\max}\left(S_p^*(\mathcal{L}_n, \mathcal{M}_m)\right) S_m.\end{aligned}$$

Since $C_{\max}\left(S_p^*\left(\mathcal{L}_n, \mathcal{M}_m\right)\right) \geq T_m = p\left(N\right)/S_m$, we deduce

$$\sum_{i=1}^{m} G_i > \left(1 - \frac{2}{m}\right) p(N). \tag{4.15}$$

If there exists a gap G_i such that $G_i \geq p_n$, then job J_n can be assigned to machine M_i and it will be completed before B. Thus, assume that for each non-zero gap G_i the inequalities $G_i < p_n \leq \frac{p(N)}{n} \leq \frac{p(N)}{m}$ hold.

If the preemption had been used before the last job is scheduled, i.e., if $J_\ell \neq J_n$, then the number of such non-zero gaps is at most $m-2$, since the assignment of job J_ℓ leaves no gap on machine $M_{\ell'}$; see Step 2(c). It follows from (4.15)

$$\left(1 - \frac{2}{m}\right) p(N) < \sum_{i=1}^{m} G_i < (m-2)\, p_n \leq (m-2)\, \frac{p\left(N\right)}{m}, \tag{4.16}$$

a contradiction.

Therefore, no preemption has been used prior to scheduling the last job n, and the number of non-zero gaps is at most $m-1$. We know that job J_n cannot be processed non-preemptively to complete by time B. Suppose that for any pair of machines M' and M'' the total gap $G' + G''$ on these two machines is less than p_n. Since each non-zero gap is less than p_n and any pair of these gaps is in total less than p_n, we deduce from (4.15) that

$$\left(1 - \frac{2}{m}\right) p(N) < \sum_{i=1}^{m} G_i \leq (m-3)\, p_n + p_n = (m-2)\, p_n,$$

leading to the same contradiction as above.

In the remaining part of this proof we show that job J_n can be processed with a single preemption, as described in Step 2(c) of the algorithm for $J_\ell = J_n$. Take a pair of machines $M_{\ell'}$ and $M_{\ell''}$ such that $G_{\ell'} + G_{\ell''} \geq p_n$. In accordance with Step 2(c), define

$$x_n = G_{\ell'},\ y_n = p_n - x_n,$$

which implies that

$$y_n + p\left(N_{\ell''}\right) = y_n + s_{\ell''}B - G_{\ell''} = p_n + s_{\ell''}B - \left(G_{\ell'} + G_{\ell''}\right) \leq s_{\ell''}B,$$

i.e., condition (4.8) holds, as required.

Assume that $p\left(N_{\ell''}\right) > 0$, which means that there is at least one job, say, J_i, assigned to $M_{\ell''}$ such that $p_i \geq p_n$. Compute

$$\frac{x_n}{s_{\ell'}} + \frac{y_n}{s_{\ell''}} \leq \frac{x_n + y_n}{s_{\ell''}} \leq \frac{p_i}{s_{\ell''}} \leq \frac{p\left(N_{\ell''}\right)}{s_{\ell''}} \leq B,$$

i.e., condition (4.9) also holds.

Now, we only need to consider the case that $p\left(N_u\right) = 0$ for all machines M_u with $u > \ell'$. Notice that for all these machines $G_u = s_u B$.

We can take the fastest machine M_1 as machine $M_{\ell'}$, since this machine has a non-zero gap prior to the assignment of job J_n; otherwise, there are at most $m-2$ non-zero gaps in schedule $S_{(1)}\left(\mathcal{L}_n, \mathcal{M}_m\right)$ and we can use (4.16) to derive a contradiction.

Thus, we only need to consider the situation that prior to the assignment of job J_n all machines, from the fastest machine M_1, have no assigned jobs. We show that job J_n can be processed with a single preemption on two fastest machines M_1 and M_2. Let us assign the last job preemptively to machines M_1 and M_2 and according to Algorithm LPT1 we make M_1 critical. For job J_n, split its processing time into two parts x_n and y_n, where $x_n = s_1 B - p\left(N_1\right)$ and that part of J_n is assigned to M_1, while the remaining part $y_n = p_n - x_n$ is assigned to M_2.

To guarantee feasibility, we need to prove that (4.9) holds, i.e.,

$$\frac{x_n}{s_1} + \frac{p_n - x_n}{s_2} = \frac{s_1 B - p\left(N_1\right)}{s_1}\left(1 - \frac{s_1}{s_2}\right) + \frac{p_n}{s_2} \leq B,$$

which after multiplying by a factor of $\frac{s_2}{s_1} > 0$, simplifies to

$$\frac{p\left(N_1\right)}{s_1}\left(1 - \frac{s_2}{s_1}\right) + \frac{p_n}{s_1} - B \leq 0. \tag{4.17}$$

We use the fact that $p_n \leq \frac{p(N)}{m}$, so that for the left hand side of (4.17) satisfies

$$\frac{p\left(N_1\right)}{s_1}\left(1 - \frac{s_2}{s_1}\right) + \frac{p_n}{s_1} - B \leq \frac{p\left(N\right)}{s_1}\left(1 - \frac{s_2}{s_1}\left(1 - \frac{1}{m}\right)\right) - B.$$

For this case,

$$\rho_m^{(1)} T_m = \frac{2\left(m-1\right)}{m} \frac{p\left(N\right)}{\sum_{i=1}^{m} s_i} \leq B,$$

so that

$$\begin{aligned}\frac{p(N)}{s_1}\left(1-\frac{s_2}{s_1}\left(1-\frac{1}{m}\right)\right)-B &\leq \frac{p(N)}{s_1}\left(1-\frac{s_2}{s_1}\left(1-\frac{1}{m}\right)\right)\\ &\quad -\frac{2(m-1)}{m}\frac{p(N)}{\sum_{i=1}^{m}s_i}.\end{aligned}$$

Since s_1 and s_2 are the two fastest machines, it follows that $\sum_{i=1}^{m} s_i \leq (s_1+(m-1)s_2)$, so that

$$\begin{aligned}&\frac{p(N)}{s_1}\left(1-\frac{s_2}{s_1}\left(1-\frac{1}{m}\right)\right)-B\\ \leq\ &\frac{p(N)}{s_1}\left(1-\frac{s_2}{s_1}\left(1-\frac{1}{m}\right)\right)-\frac{2(m-1)}{m}\frac{p(N)}{s_1+(m-1)s_2}\\ =\ &-\frac{p(N)}{ms_1^2(s_1+(m-1)s_2)}\left((m-2)s_1^2-(m-1)^2s_1s_2+(m-1)^2s_2^2\right).\end{aligned}$$

To show that (4.17) holds, we demonstrate that

$$(m-2)s_1^2-(m-1)^2s_1s_2+(m-1)^2s_2^2\geq 0. \tag{4.18}$$

Recall that Algorithm LPT1 assigns all jobs except job J_n to machine M_1, so that these jobs complete at time $(p(N)-p_n)/s_1$. Since $p_n \leq \frac{1}{m}p(N)$, machine M_1 completes no later than time $\frac{m-1}{m}\frac{p(N)}{s_1}$. On other other hand, job J_n cannot be assigned to M_2 or to any slower machine to be processed without preemption and to complete by time B, so that $\frac{p_n}{s_2}>B$. Thus, we have that

$$\frac{m-1}{m}\frac{p(N)}{s_1}\leq B<\frac{p_n}{s_2}\leq\frac{p(N)}{ms_2},$$

implies that $s_1\geq(m-1)s_2$. Then we can express the ratio of the machine speeds as $\frac{s_1}{s_2}=(m-1)\alpha$, where $\alpha\geq 1$. Substituting into (4.18), we deduce

$$\begin{aligned}&(m-2)s_1^2-(m-1)^2s_1s_2+(m-1)^2s_2^2\\ =\ &(m-2)(m-1)^2\alpha^2s_2^2-(m-1)^3\alpha s_2^2+(m-1)^2s_2^2\\ =\ &(m-1)^2s_2^2\left((m-2)\alpha^2-(m-1)\alpha+1\right)\\ =\ &(m-1)^2s_2^2((m-2)\alpha-1)(\alpha-1)\geq 0,\end{aligned}$$

where the last inequality holds since $\alpha\geq 1$ and we are only interested in $m\geq 3$. Thus, Algorithm LPT1 creates a schedule with a simple preemption in which both machines complete by time B.

Having considered all possible cases, we conclude that the minimal counterexample does not exist and $C_{\max}\left(S_{(1)}\left(\mathcal{L}_n, \mathcal{M}_m\right)\right) \leq B$ for all instances as required. ■

To see that the established bound is tight, consider an instance of the problem with m machines, all of unit speed except machine M_1, which has speed $m-1$, and m jobs of unit length, i.e., $\mathcal{M}_m = (m-1, 1, 1, \ldots, 1)$ and $\mathcal{L}_n = (1, 1, \ldots, 1)$. Clearly,

$$C_{\max}\left(S_p^*\left(\mathcal{L}_n, \mathcal{M}_m\right)\right) = T_m = \frac{m}{2m-2}.$$

For this instance, an optimal preemptive schedule requires $2(m-1)$ preemptions, which is the maximum estimate due to Gonzalez and Sahni (1978).

Algorithm LPT1 will assign at most one job to each of k of the machines $M_2, \ldots, M_m$, where $1 \leq k \leq m-1$, while the remaining $m-k$ jobs are assigned to machine M_1, so that $C_{\max}\left(S_{(1)}\left(\mathcal{L}_n, \mathcal{M}_m\right)\right) = 1$. Notice that schedule $S_{(1)}\left(\mathcal{L}_n, \mathcal{M}_m\right)$ is an optimal non-preemptive schedule and its makespan cannot be reduced if a single preemption is allowed. □

Note that Theorem 4.2 still holds if a looser version of Algorithm Qm(1)LPT is used. It is sufficient for the preemption to be used in Step 2(c) at *any* time, as long as any machine of the chosen pair is made critical by the preempted job and the preemption is feasible.

Moreover, note that if Algorithm Qm(1)LPT is modified so that it is run allowing no preemption and B is redefined as $\rho_m^{(0)} C_{\max}\left(S_p^*\right)$, where $\rho_m^{(0)} = 2-1/m$, then the proof of Theorem 4.2 can be appropriately adjusted to establish that $\rho_m^{(0)} = 2-1/m$ is the power of preemption. This is achieved by using a less restrictive algorithm than those used for the same purpose in Woeginger (2000) and Soper and Strusevich (2014b).

4.2.4 Parametric Analysis of the Power of Limited Preemption for Three Uniform Machines

In this section we consider the problem of minimizing the makespan with limited preemption on three uniform machines. The analysis of the power of limited preemption performed in this section follows the classification of instances for the problem on three uniform machines by Soper and Strusevich (2014a), as defined in Section 2.2.1.

Consider the instances of this problem that belong to Classes 1 and 2 of this problem, as defined by Soper and Strusevich (2014a). An instance of this problem is defined by the list of processing times $\mathcal{L}_n$, and a list of machines $\mathcal{M}_3 = \{M_1, M_2, M_3\}$. We assume that in list $\mathcal{L}_n$, jobs are numbered in such a way that

$$p_1 = \max\{p_j | j \in N\}, \ p_2 = \max\{p_j | j \in N \backslash \{J_1\}\},$$

while the remaining jobs are numbered arbitrarily.

For an instance $I = (\mathcal{L}_n, \mathcal{M}_3)$ of Class r, $1 \le r \le 2$, let $\rho_3^{(1)}$ (Class r) denote the power of a single preemption for instances of that class. We derive a tight upper bound on ρ_3 (Class r) as a function of the machine speeds $s_1 \ge s_2 \ge s_3$.

Algorithm Q3(1)C[1&2]

INPUT: A Class 2 or 3 instance $I = (\mathcal{L}_n, \mathcal{M}_3)$ for problem $Q3\,|\#pmtn \le 1|\, C_{\max}$

OUTPUT: A 1-preemptive schedule $S_{(1)}(I)$.

Step 1. If I belongs to Class 1, go to Step 2; otherwise, go to Step 3.

Step 2. Find schedule $S_{(1)}^H((\mathcal{L}_n, \mathcal{M}_3))$ in which job J_1 is assigned to be processed on machine M_1, while the remaining jobs are scheduled by Algorithm Q2(1) on machines M_2 and M_3. Go to Step 4.

Step 3. Find schedule $S_{(1)}^H((\mathcal{L}_n, \mathcal{M}_3))$ in which jobs J_1 and J_2 are scheduled by Algorithm Q2(1) on machines M_1 and M_2, while the remaining jobs are processed non-preemptively in any order on machine M_3. Go to Step 4.

Step 4. Stop.

Algorithm Q3(1)C[1&2] is analyzed below. Part of the analysis is based on the parametric bound (4.6) developed in Jiang et al. (2014) for two uniform machines.

Theorem 4.3. *For an instance $I = (\mathcal{L}_n, \mathcal{M}_3)$ of Class r, $1 \le r \le 2$, Algorithm Q3(1)C[1&2] takes $O(n)$ time and finds a schedule $S_{(1)}^H((\mathcal{L}_n, \mathcal{M}_3))$ such that*

$$\rho_3^{(1)}(\text{Class } r) = \frac{C_{\max}\left(S_{(1)}^*(\mathcal{L}_n, \mathcal{M}_3)\right)}{C_{\max}\left(S_p^*(\mathcal{L}_n, \mathcal{M}_3)\right)} \le \frac{C_{\max}\left(S_{(1)}^H(\mathcal{L}_n, \mathcal{M}_3)\right)}{C_{\max}\left(S_p^*(\mathcal{L}_n, \mathcal{M}_3)\right)} \le \Phi(s_1, s_2, s_3), \tag{4.19}$$

where

$$\Phi(s_1, s_2, s_3) = \left\{ \begin{array}{ll} \frac{2s_2^2 + s_2 s_3 - s_3^2}{2s_2^2}, & \text{if } r = 1 \\ \frac{2s_1^2 + s_1 s_2 - s_2^2}{2s_1^2}, & \text{if } r = 2 \end{array} \right\}.$$

The first bound is tight for two fast machines and two slow machines, viz $s_1 = s_2 = s \ge s_3$, and $s_1 = s \ge s_2 = s_3$, while the second is tight for all values of the machine speeds.

Proof: Initially, observe that the running time of Algorithm Q3(1)C[1&2] is determined by the running time of Algorithm Q2(1), and, therefore is linear in n.

For a given instance of Class r, $1 \le r \le 2$, define the lists $\mathcal{L}_r$ and $\mathcal{M}_r$, that contain the r longest jobs and the r fastest machine, respectively. Also define the lists $\mathcal{L}'_r$ and $\mathcal{M}'_r$ obtained from the lists $\mathcal{L}_n$ and $\mathcal{M}_3$ by the removal of the r longest jobs and the r fastest machines, respectively. In other words, $\mathcal{L}'_r = (p_{r+1}, \ldots, p_n)$ and $\mathcal{M}'_r = (s_{r+1}, \ldots, s_3)$.

By definition of a Class r instance, we have that

$$C_{\max}\left(S_p^*\left(\mathcal{L}_r, \mathcal{M}_r\right)\right) \ge C_{\max}\left(S_p^*\left(\mathcal{L}_n, \mathcal{M}_3\right)\right),$$

which implies

$$C_{\max}\left(S_p^*\left(\mathcal{L}'_r, \mathcal{M}'_r\right)\right) \le C_{\max}\left(S_p^*\left(\mathcal{L}_n, \mathcal{M}_3\right)\right) \le C_{\max}\left(S_p^*\left(\mathcal{L}_r, \mathcal{M}_r\right)\right).$$

For schedule $S^H_{(1)}\left(\mathcal{L}_n, \mathcal{M}_3\right)$ found by Algorithm Q3(1)C[1&2], we have that

$$C_{\max}\left(S^H_{(1)}\left(\mathcal{L}_n, \mathcal{M}_3\right)\right) = \max\left\{C_{\max}\left(S^H_{(1)}\left(\mathcal{L}_r, \mathcal{M}_r\right)\right), C_{\max}\left(S^H_{(1)}\left(\mathcal{L}'_r, \mathcal{M}'_r\right)\right)\right\}.$$

It follows that

$$\begin{aligned}
\rho_3^{(1)}\left(\text{Class } r\right) &\le \frac{C_{\max}\left(S^*_{(1)}\left(\mathcal{L}_n, \mathcal{M}_3\right)\right)}{C_{\max}\left(S_p^*\left(\mathcal{L}_n, \mathcal{M}_3\right)\right)} = \frac{C_{\max}\left(S^*_{(1)}\left(\mathcal{L}_n, \mathcal{M}_3\right)\right)}{C_{\max}\left(S_p^*\left(\mathcal{L}_r, \mathcal{M}_r\right)\right)} \\
&\le \frac{\max\left\{C_{\max}\left(S^H_{(1)}\left(\mathcal{L}_r, \mathcal{M}_r\right)\right), C_{\max}\left(S^H_{(1)}\left(\mathcal{L}'_r, \mathcal{M}'_r\right)\right)\right\}}{C_{\max}\left(S_p^*\left(\mathcal{L}_r, \mathcal{M}_r\right)\right)} \qquad (4.20) \\
&\le \max\left\{\frac{C_{\max}\left(S^H_{(1)}\left(\mathcal{L}_r, \mathcal{M}_r\right)\right)}{C_{\max}\left(S_p^*\left(\mathcal{L}_r, \mathcal{M}_r\right)\right)}, \frac{C_{\max}\left(S^H_{(1)}\left(\mathcal{L}'_r, \mathcal{M}'_r\right)\right)}{C_{\max}\left(S_p^*\left(\mathcal{L}'_r, \mathcal{M}'_r\right)\right)}\right\}.
\end{aligned}$$

For Class 1, Step 2 of Algorithm Q3(1)C[1&2] guarantees that

$$C_{\max}\left(S^H_{(1)}\left(\mathcal{L}_1, \mathcal{M}_1\right)\right) = p_1/s_1,$$

i.e.,

$$\frac{C_{\max}\left(S^H_{(1)}\left(\mathcal{L}_1, \mathcal{M}_1\right)\right)}{C_{\max}\left(S_p^*\left(\mathcal{L}_n, \mathcal{M}_3\right)\right)} = 1.$$

Since the remaining jobs are scheduled on machines M_2 and M_3 optimally with at most one preemption, we have that

$$C_{\max}\left(S^H_{(1)}\left(\mathcal{L}_1^{'},\mathcal{M}_1^{'}\right)\right)=C_{\max}\left(S^*_{(1)}\left(\mathcal{L}_1^{'},\mathcal{M}_1^{'}\right)\right),$$

so that

$$\frac{C_{\max}\left(S^H_{(1)}\left(\mathcal{L}_1^{'},\mathcal{M}_1^{'}\right)\right)}{C_{\max}\left(S^*_p\left(\mathcal{L}_1^{'},\mathcal{M}_1^{'}\right)\right)}=\frac{C_{\max}\left(S^*_{(1)}\left(\mathcal{L}_1^{'},\mathcal{M}_1^{'}\right)\right)}{C_{\max}\left(S^*_p\left(\mathcal{L}_1^{'},\mathcal{M}_1^{'}\right)\right)}\le\frac{2s_2^2+s_2s_3-s_3^2}{2s_2^2}$$

where the last inequality follows from (4.6) applied to the machines with speeds s_2 and s_3. It is the power of preemption for two uniform machines derived by Jiang et al. (2014).

A tight instance for $r=1$ is given by a set of three jobs such that $\mathcal{L}_3=\left(p,p\frac{s_2+s_3}{2s_1},p\frac{s_2+s_3}{2s_1}\right)$ and $\mathcal{M}_3=(s_1,s_2,s_3)$. It follows that

$$C_{\max}\left(S^*_p\left(\mathcal{L}_3,\mathcal{M}_3\right)\right)=C_{\max}\left(S^*_p\left(\mathcal{L}_1,\mathcal{M}_1\right)\right)=\frac{p_1}{s_1}=\frac{p}{s_1}=C_{\max}\left(S^*_p\left(\mathcal{L}_1^{'},\mathcal{M}_1^{'}\right)\right),$$

hence

$$\rho_3^{(1)}=\frac{2s_2^2+s_2s_3-s_3^2}{2s_2^2},$$

since according to Jiang et al. (2014) the tight example on two uniform machines is given by two jobs of equal length,.where machines are numbered according to (2.14).

For Class2. $r=2$, and $C_{\max}\left(S^*_p\left(\mathcal{L}_n,\mathcal{M}_3\right)\right)=C_{\max}\left(S^*_p\left(\mathcal{L}_2,\mathcal{M}_2\right)\right)$. Considering the second term in (4.20). In this case, we have

$$C_{\max}\left(S^*_p\left(\mathcal{L}_2^{'},\mathcal{M}_2^{'}\right)\right)=C_{\max}\left(S^H_p\left(\mathcal{L}_2^{'},\mathcal{M}_2^{'}\right)\right)$$

since the machine environment $\mathcal{M}_2^{'}$ consists only of of machine M_3, thus

$$\frac{C_{\max}\left(S^*_{(1)}\left(\mathcal{L}_2^{'},\mathcal{M}_2^{'}\right)\right)}{C_{\max}\left(S^*_p\left(\mathcal{L}_2^{'},\mathcal{M}_2^{'}\right)\right)}=1$$

.To complete the heuristic schedule we optimally assign the two longest jobs on machines M_1 and M_2 according to AlgorithmQ2(1) using the single preemption, when

$$\frac{C_{\max}\left(S^h_{(1)}\left(\mathcal{L}_2,\mathcal{M}_2\right)\right)}{C_{\max}\left(S^*_p\left(\mathcal{L}_2,\mathcal{M}_2\right)\right)}=\frac{C_{\max}\left(S^*_{(1)}\left(\mathcal{L}_2,\mathcal{M}_2\right)\right)}{C_{\max}\left(S^*_p\left(\mathcal{L}_2,\mathcal{M}_2\right)\right)}\le\left(2s_1^2+s_1s_2-s_2^2\right)/2s_1^2.$$

Hence $\rho_3^{(1)}\le\left(2s_1^2+s_1s_2-s_2^2\right)/2s_1^2$ for this class. A tight instance is given by a set of 3 jobs with $\mathcal{L}_3 = \left(p, p, p\frac{2s_3}{s_1+s_2}\right)$ and $\mathcal{M}_3 = (s_1, s_2, s_3)$. It follows that $C_{\max}\left(S_p^*\left(\mathcal{L}_3, \mathcal{M}_3\right)\right) = C_{\max}\left(S_p^*\left(\mathcal{L}_2, \mathcal{M}_2\right)\right) = 2p/\left(s_1+s_2\right)$, while schedule $S_{(1)}^*\left(\mathcal{L}_3, \mathcal{M}_3\right)$ is obtained by scheduling the identical jobs J_1 and J_2 on machines M_1 and M_2 by Algorithm Q2Pr1, with the remaining job assigned to machine M_3 $\square$

The parametric analysis of the third class of instances is more intricate and is not performed as a part of this thesis. We note that for that class, two cases arise depending on the number of faster machines in the instance.

4.3 Unrelated Machines

In this section we consider the problem of minimizing the makespan in limited-preemptive schedules on unrelated machines. In particular, for the problem of minimizing the makespan on 2 unrelated machines, we show that the problem of obtaining a limited preemptive schedule is NP-hard. We prove this statement in the following lemma.

Lemma 4.2. *Problem $R2\left|\#pmnt \le 1\right| C_{\max}$ is NP-hard in the ordinary sense.*

Proof: We provide reduction from the PARTITION problem described in Section 1.2.

Given an instance of PARTITION:, define the following instance of a decision version of problem $R2\left|\#pmnt = 1\right| C_{\max}$ with two machines A and B and the set $N = \{1, 2, \ldots, n\}$ of jobs:

$$\begin{aligned} n &= r+2; \\ a_j &= 2e_j,\ b_j = e_j,\ j \in R; \\ a_{r+1} &= 11E,\ b_{r+1} = 2E; \\ a_{r+2} &= 6E,\ b_{r+2} = 4E. \end{aligned}$$

It is required to verify whether there exists a schedule $S^{(1)}$ with a single preemption with $C_{\max}\left(S^{(1)}\right) \le 5E$.

Suppose that PARTITION has a solution and R_1 and R_2 are the required subsets. Then assign an arbitrary sequence of jobs $j \in R_1$ to be processed on machine A in the time interval $[0, 2E]$, and an arbitrary sequence of jobs $j \in R_2$ and job $r+1$ to be processed on machine B schedule in the time interval $[2E, 5E]$. Job $r+2$ is processed with preemption, in the time interval $[0, 2E]$ on machine B and in the time interval $[2E, 3E]$ on machine B. For the resulting schedule S_1 both machines complete there processing at time $5E$. The preemptive processing of job $r+2$ is feasible, since

$$\frac{2E}{4E}+\frac{3E}{6E}=1.$$

Suppose now that schedule $S^{(1)}$ exists. First notice that in $S^{(1)}$ either job $r+1$ or job $r+2$ must be processed with preemption. Indeed, in order to complete by time $5E$ none of these jobs can be processed without preemption on machine A, and if both are assigned to be processed non-preemptively on machine B they will complete no earlier than time $6E$.

Suppose job $r+2$ is processed without preemption, and job $r+2$ with preemption. To complete by time $5E$, job $r+2$ must be assigned to machine B. Then job $r+1$ can be processed on B for at most E time units, which is at most 50% of its overall processing. No less than 50% of job $r+1$ must be performed on machine A, which makes at least $5.5E$ units of processing.

Thus, in schedule S_1 job $r+2$ is processed with preemption and job $r+2$ is processed on machine B.

Suppose that in S_1 job $r+2$ is processed on machine A for $3E+x$ time units and on machine B for $2E-y$ time units, where x and y are both positive. To complete this job by time time $5E$ we must have that $x \leq y$. On the other hand, for feasible processing of job $r+2$ the equality

$$\frac{2E-y}{4E}+\frac{3E+x}{6E}=1$$

must hold, which implies that $y=\frac{2}{3}x<x$; a contradiction.

Now suppose that in S_1 job $r+2$ is processed on machine A for $3E-x$ time units and on machine B for $2E+y$ time units, where x and y are both positive. To complete this job by time time $5E$ we must have that $x \geq y$. For feasible processing of job $r+2$ the equality

$$\frac{2E+y}{4E}+\frac{3E-x}{6E}=1$$

must hold, which implies that $y=\frac{2}{3}x$.

Let N_A and N_B denote the subsets of jobs of set R assigned to be processed in schedule S_1 on machine A and B, respectively. Then for machine B to complete its jobs by time $5E$ the inequality

$$\left(2E+\frac{2x}{3}\right)+2E+e\left(N_B\right)\leq 5E,$$

which implies that $e(N_B) \leq E - \frac{2x}{3}$, so that $e(N_A) \geq E + \frac{2x}{3}$. Then machine A completes its jobs no earlier than time

$$2\left(E + \frac{2x}{3}\right) + 3E - x = 5E + \frac{x}{3}.$$

We deduce that for schedule S_1 to exist, job $r+2$ must be processed on machine B for $2E$ time units and on machine A for $3E$ time units. This implies that in S_1, $b(N_B) = e(N_B) = E$, i.e., the sets N_A and N_B form a solution to Partition. $\square$

4.3.1 Single Preemption on two Unrelated Machines

Consider the problem of minimizing the makespan with a single preemption on two unrelated machines. In the preemptive version of this problem, denoted by $R2\,|pmtn|\,C_{\max}$, an optimal schedule is known to require at most 2 preemptions. Consider an instance I_ℓ, where an optimal preemptive schedule $S_p^*(I_\ell)$ requires at most one preemption. As only one preemption is required, instance I_ℓ is said to belong to Class-ℓ instances for problem $R2\,|\#pmtn \leq 1|\,C_{\max}$ for which

$$C_{\max}\left(S_p^*(I_\ell)\right) = C_{\max}\left(S_{(1)}^*(I_\ell)\right).$$

In this section we determine the power of preemption for instances of Class ℓ. For this problem, a set of n jobs is to be scheduled on two unrelated machines M_1 and M_2, so that the in the optimal preemptive schedule $S_p^*(I_\ell)$ there is only one job which is preempted between the two machines. We denote this job by J_k.

In a similar approach to the one detailed for the case of two unrelated machines in Section 2.3.2, we construct the non-preemptive schedule by selecting the best non-preemptive schedule resulting from rounding the result of the LP solution to the preemptive problem.

Assuming that all other jobs have been optimally scheduled on the machines, a rounding of the LP relaxation results in two possible schedules S'_{np}, where J_k is entirely processed by M_1 , and S''_{np}, where J_k is processed entirely by M_2. Following from Section 2.3.2, let x_{ij}, $x_{ij} \geq 0$, denote the fraction of job J_j processed on machine M_i, so that

$$\sum_{i=1}^{m} x_{ij} = 1,\ 1 \leq j \leq n$$

In the preemptive schedule $S_p^*(I_\ell)$, job J_k is processed for $x_{1k}p_{1k}$ amount of time on machine M_1, and $x_{2k}p_{2k}$ amount of time on machine M_2. Recall that in a feasible schedule, a job may not be processed by more than one machine at a time. Thus, the total amount of time that the splitting job J_k is processed on machines M_1 and M_2 cannot exceed the total duration of the schedule denoted by C, as the alternative implies that there is a clash of the processing intervals of J_k on the two machines. Hence we determine that

$$x_{1k}p_{1k} + x_{2k}p_{2k} \leq C.$$

In the rounded, non-preemptive solution, job J_k is entirely on one of the machines, while the assignment of any remaining jobs is not affected. Thus, if J_k is scheduled to be processed entirely by machine M_1 in a non-preemptive schedule S'_{np}, the total processing time of M_1 will be increased by $x_{2k}p_{1k}$, or alternatively, in a schedule S''_{np} where J_k is processed by M_2, the total processing time is increased by $x_{1k}p_{2k}$.

A tight instance for this class can be obtained by solving the following linear programming formulation

$$\begin{aligned}
&\text{Maximize:} \quad \min\{x_{2k}p_{1k}, x_{1k}p_{2k}\} \\
&\text{Subject to:} \\
&\qquad x_{1k} + x_{2k} = 1 \\
&\qquad x_{1k}p_{1k} + x_{2k}p_{2k} \leq C \\
&\qquad x_{ik} \geq 0 \qquad\qquad , \text{for} \quad 1 \leq i \leq 2
\end{aligned}$$

Hence, we obtain

$$\min\{x_{2k}p_{1k}, x_{1k}p_{2k}\} \leq \min\left\{x_{2k}p_{1k}, (C - x_{2k}p_{2k})\frac{p_{2k}}{p_{1k}}\right\},$$

where the right-hand side takes the smallest value if

$$x_{2k}p_{1k} = (C - x_{2k}p_{2k})\frac{p_{2k}}{p_{1k}}.$$

Note that in this case we are only concerned with the assignment of J_k to one of the machines, as all other jobs have been optimally assigned. Hence, as we are concerned with the different processing times of a single job, it is possible to consider this problem as a uniform machine problem by expressing the processing time of J_k on machine M_2 as a factor of p_{1k} and *"machine speed"* s, so that

$$p_{2k} = s \cdot p_{1k}.$$

Thus, we obtain

$$\begin{aligned} x_{2k}p_{1k} &= \left(C - x_{2k}p_{1k}s\right)s \\ &= \frac{Cs}{1+s^2}. \end{aligned}$$

which takes a maximum value of $C/2$ when $s = 1$. Hence, we obtain

$$\begin{aligned} \frac{C_{\max}\left(S_{np}^{H}\left(I_\ell\right)\right)}{C_{\max}\left(S_{p}^{*}\left(I_\ell\right)\right)} &\leq \frac{C + C/2}{C} \\ &= \frac{3}{2}. \end{aligned}$$

A tight instance with three jobs J_1, $J_{2,}J_3$, with the following processing requirements:

#	p_{i1}	p_{i2}	p_{i3}
M_1	1	$+\infty$	1
M_2	$+\infty$	1	1

.

CHAPTER 5

Power of Splitting

5.1 Review of Job Splitting

The parallel machine scheduling problems considered in previous chapters follow the assumption that a job may be processed by only one machine at any given time. In this chapter we consider a particular type of parallel machine scheduling where that particular constraint is removed.

When the processing requirement is considered as a total demand of a product in production planning, it may be arbitrarily split on the machines in order to finish the processing of all demands as soon as possible. This process is referred to by Potts and Wassenhove (1992) as *lotstreaming* or *lotsizing* and the split parts as *continuous sublots*. A problem which arises from the processing of different types of fabric in the textile industry is studied by Serafini (1996), who provides some polynomial-time algorithms for minimizing the *maximum tardiness* and the *maximum weighted tardiness* objectives, where each job may be split arbitrarily and processed independently by the available machines.

For parallel machine scheduling problems with splitting, Xing and Zhang (2000) prove that if no setup times are present, the problems of minimizing the makespan and total flow time on identical, uniform and unrelated machines, are solvable in polynomial time. Specifically, for problems on identical and uniform machines, they show that an optimal schedule is obtained by splitting each job into m sublots, and assigning these sublots to each of the m machines in a given instance, so that all sublots of a given job begin processing at the same time on all machines.

The results by Xing and Zhang (2000) which are essential for the work in this chapter are presented in the following lemmas.

Lemma 5.1 (Xing and Zhang (2000)). *For the problems of minimizing the makespan and sum of completion times, there exists an optimal schedule such that m identical, uniform, or unrelated machines, complete at the same time.*

Lemma 5.1 yields the following results. For identical machines, the makespan of an optimal splitting schedule S^*_{split} is given by

$$C_{\max}\left(S^*_{split}\right) = \frac{p(N)}{m}. \tag{5.1}$$

For uniform machines, the corresponding makespan of an optimal schedule is given by

$$C_{\max}\left(S^*_{split}\right) = \frac{p(N)}{s(\mathcal{M})}, \tag{5.2}$$

where

$$s(\mathcal{M}) = \sum_{i=1}^{m} s_i.$$

The following lemma shows that any scheduling problem with splitting jobs on identical and uniform machines, can be converted into a single machine scheduling problem.

Lemma 5.2 (Xing and Zhang (2000)). *For the problems of minimizing the makespan and total completion time on identical and uniform machines, there exists an optimal schedule, such that each job is split into m sublots, and these m sublots are assigned to m machines, so that they begin processing and complete at the same time on each of the m machines.*

Based on the above Lemmas, Xing and Zhang (2000) prove that, without setup times, the problems of minimizing the makespan and sum of completion times, and a number of other objective functions on identical and uniform machines, are polynomially solvable. This is formally described in the following theorem.

Theorem 5.1 (Xing and Zhang (2000)). *There are polynomial algorithms for scheduling problems of minimizing the makespan and sum of completion times on identical and unrelated machines with splitting jobs and no setup times.*

Furthermore, they show that an optimal splitting schedule on unrelated machines is obtained by solving the following linear programming formulation

$$\begin{aligned}
&\text{Minimize} \quad C_{\max} \\
&\text{Subject to:} \\
&\qquad \sum_{i=1}^{m} \frac{t_{ij}}{p_{ij}} = 1 && \text{, for} \quad 1 \le j \le n \\
&\qquad \sum_{j=1}^{n} t_{ij} = C_{\max} && \text{, for} \quad 1 \le i \le m \\
&\qquad t_{ij} \ge 0 && \text{, for} \quad 1 \le i \le m,\ 1 \le j \le n
\end{aligned}$$

In the above formulation, t_{ij} represents the total amount of time that job J_j spends on machine M_i.

Following the results by Xing and Zhang (2000), for the objectives of minimizing the makespan and total completion time on identical, uniform and unrelated machines, we perform an analysis of the *power of splitting.*

Similarly to the power of preemption, for some objective function Φ, we define the power of splitting as the supremum of the ratio $\Phi\left(S^*_{np}\right)/\Phi(S^*_{split})$ across all instances of a given problem. For a problem on m parallel machines, we represent the power of splitting by τ_m, so that

$$\frac{\Phi\left(S^*_{np}\right)}{\Phi(S^*_{split})} \le \tau_m. \tag{5.3}$$

Moreover, we denote an optimal schedule with splitting jobs by S^*_{split}.

5.2 Identical Machines

5.2.1 Makespan

Consider the problem of minimizing the makespan on m identical machines. Given a problem instance I, we denote by λ the ratio of the makespan of a preemptive schedule S^*_p over the makespan of a splitting schedule S^*_{split}, so that

$$\lambda = \frac{C_{\max}\left(S^*_p\right)}{C_{\max}\left(S^*_{split}\right)}. \tag{5.4}$$

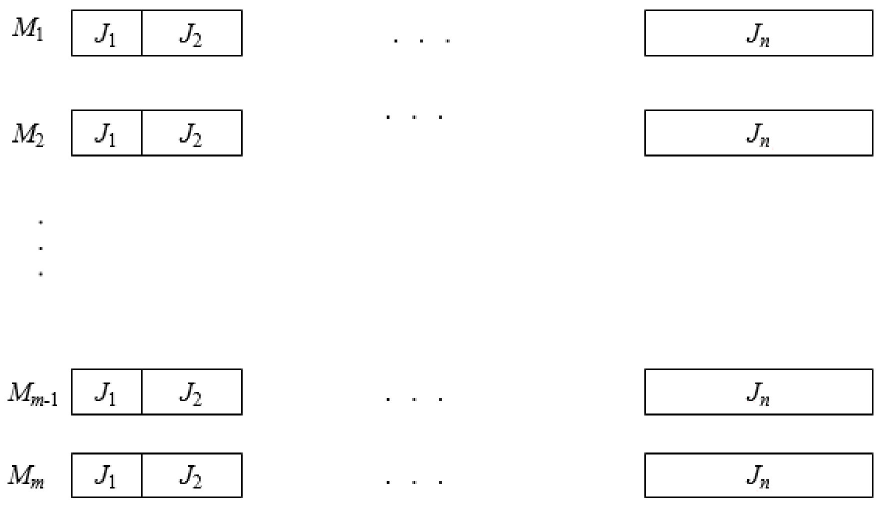

Figure 5.1: A split schedule on m parallel idenitcal machines

Recall from (5.1) that the makespan of an optimal splitting schedule on m identical machines is given by the average machine load; see Figure 5.1. We identify the following classes of instances, based on the value of λ.

Class 1 Instances

We define Class 1 as the set of instances for which the makespan of an optimal preemptive schedule is given by the length of the largest job, so that

$$C_{\max}\left(S_p^*\right) = p_1.$$

For this class, the value of λ is given by

$$\begin{aligned}\lambda &= \frac{p_1}{\frac{p(N)}{m}} \\ &= m\frac{p_1}{p\left(N\right)}\end{aligned}$$

it holds that $\lambda > 1$. For instances of this class we further define the following cases:

Case 1.1

If there are at most as many jobs as machines, so that $n \leq m$, an optimal assignment of jobs to machines is achieved by scheduling exactly one job to every machine in the schedule. Hence, it is easy to see that for a non-preemptive schedule S_{np}^*, we obtain

$$C_{\max}\left(S_{np}^*\right) = C_{\max}\left(S_p^*\right) = p_1.$$

The power of splitting in this case is given by

$$\tau_m = \frac{p_1}{\frac{p(N)}{m}} = \lambda. \tag{5.5}$$

Case 1.2

For Class 1 instances where the number of jobs exceeds the number of machines, so that $n > m$, given the smallest index g, $0 \leq g \leq m-1$, for which (3.4) holds, Algorithm P-nonpreemptive described in Section 3.1 finds a schedule S_{np}, so that

$$C_{\max}(S_{np}) \leq \left(2 - \frac{2}{m-g+1}\right) \frac{p(N) - \sum_{j=1}^{g} p_j}{m-g}. \tag{5.6}$$

Lemma 5.3. *Given an instance on m identical machines, if $n > m$ and $\lambda > 1$, then there is a g, $g \geq 1$ for which (3.4) holds.*

Proof: This proof is by contradiction.

Suppose initially that the statement does not hold, so that for $n > m$ and $\lambda > 1$ there is a $g \geq 1$, such that

$$p_{g+1} > \frac{p(N) - \sum_{j=1}^{g} p_j}{m-g}. \tag{5.7}$$

For $g = m-1$, we obtain

$$\begin{aligned} p_m &> \frac{p(N) - \sum_{j=1}^{m-1} p_j}{m-(m-1)} \\ &= p(N) - \sum_{j=1}^{m-1} p_j. \end{aligned} \tag{5.8}$$

It is apparent that (5.8) does not hold, as this implies

$$p_m > p_m + p_{m+1} + \ldots + p_n,$$

which clearly is impossible, as $m > n$, and all jobs have positive, non-zero processing times. Therefore, the initial statement holds. $\square$

Lemma 5.4. *Given an instance on m identical machines, where $n > m$, $\lambda > 1$, and*

$g \geq 1$, *then the inequality*

$$\frac{p(N) - \sum_{j=1}^{g} p_j}{m-g} \leq \frac{p(N)}{m}, \tag{5.9}$$

holds for all g, $1 \leq g \leq m-1$.

Proof:

This proof is by induction. For our base case consider $g = 1$, so that by replacing the value of g in (5.9), we obtain

$$\frac{p(N) - p_1}{m-1} \leq \frac{p(N)}{m} \tag{5.10}$$

As per the definition of the class, it is known that

$$p_1 > \frac{p(N)}{m},$$

hence, for the left hand side of (5.10) we obtain

$$\begin{aligned} \frac{p(N) - p_1}{m-1} &\leq \frac{p(N) - \frac{p(N)}{m}}{m-1} \\ &= \frac{p(N)}{m} \end{aligned}$$

which satisfies (5.9).

For the next part of this proof, given an index q, $1 \leq q \leq g$, suppose that

$$\frac{p(N) - \sum_{j=1}^{q-1} p_j}{m-q+1} \leq \frac{p(N)}{m}.$$

For $g = (q+1)$, we obtain

$$\begin{aligned} \frac{p(N) - \sum_{j=1}^{(q+1)-1} p_j}{m-(q+1)+1} &= \frac{p(N) - \sum_{j=1}^{q} p_j}{m-q} \\ &= \frac{p(N) - \sum_{j=1}^{q-1} p_j - p_q}{m-q} \end{aligned} \tag{5.11}$$

By definition of g, the inequality

$$p_g > \frac{p(N) - \sum_{j=1}^{g-1} p_j}{m-g+1},$$

holds, so that for (5.11) we obtain

$$\begin{aligned}\frac{p(N)-\sum_{j=1}^{q-1}p_j-p_q}{m-q} &\leq \frac{p(N)-\sum_{j=1}^{q-1}p_j-\left(\frac{p(N)-\sum_{j=1}^{g-1}p_j}{m-g+1}\right)}{m-q}\\ &= \frac{\left(p(N)-\sum_{j=1}^{q-1}p_j\right)(m-q)}{(m-q)(m-g+1)}\\ &= \frac{p(N)-\sum_{j=1}^{q-1}p_j}{m-q+1},\end{aligned}$$

thus the assumption holds. □

From Lemma 5.3 and Lemma 5.4, with respect to (5.6), we determine

$$\begin{aligned}C_{\max}(S_{np}) &\leq \left(2-\frac{2}{m-g+1}\right)\frac{p(N)-\sum_{j=1}^{g}p_j}{m-g}\\ &\leq \left(2-\frac{2}{m-g+1}\right)\frac{p(N)}{m}\\ &= \left(2-\frac{2}{m-g+1}\right)C_{\max}\left(S^*_{split}\right).\end{aligned}$$

Hence,

$$\tau_m \leq 2-\frac{2}{m-g+1}. \tag{5.12}$$

Class 2 Instances

This class is defined as the set of all instances on identical machines where the makespan of the optimal preemptive schedule S_p^* is given by the average machine load, which coincides with the makespan of an optimal splitting schedule S^*_{split}, so that

$$C_{\max}\left(S_p^*\right)=C_{\max}\left(S^*_{split}\right)=T_m,$$

from which we determine $\lambda > 1$. Due to this, the bounds for the power of splitting coincide with that of the power of preemption for instances of this class, so that

$$\tau_m=\rho_m\leq 2-\frac{2}{m+1}. \tag{5.13}$$

For all instances of the identical machine problem, the following algorithm finds a non-preemptive schedule S_{np}, which satisfies the above bounds.

Algorithm P-split-np

INPUT: An instance I of problem $Pm\,|\,|\,C_{\max}$.

OUTPUT: A non-preemptive schedule S^{*}_{np}.

Step 1. Compute $\lambda = C_{\max}\left(S^{*}_{p}\right)/C_{\max}\left(S^{*}_{split}\right)$.

Step 2. If $\lambda > 1$ and $n \leq m$, run Algorithm *LS* for instance I.

Step 3. If $\lambda > 1$ and $n > m$, run algorithm *P-nonpreemptive* for instance I.

Step 4. If $\lambda = 1$, run algorithm *P-nonpreemptive* for instance I.

Step 5. Stop.

Algorithm *P-split-np* runs in $O(nm)$ time, assuming that jobs are numbered in LPT order. In the following theorem we prove that *P-split-np* finds the required non-preemptive schedules for which, bounds (5.5), (5.12) and (5.13) hold, and also demonstrate the tighness of these bounds.

Theorem 5.2. *Algorithm P-split-np finds a required non-preemptive schedule, for which bounds (5.5), (5.12) and (5.13) on τ_m hold for respective classes on instances, and furthermore these bounds are tight.*

Proof:

Consider an instance I_1 of Class 1 where $n \leq m$. The algorithm finds the required schedule at Step 2 by running the list scheduling algorithm for I_1. In this case, Algorithm LS assigns exactly one job to each machine, and obtains an optimal non-preemptive schedule $S^{*}_{np}(I_1)$ with makespan

$$C_{\max}\left(S^{*}_{np}(I_1)\right) = p_1,$$

where p_1 is the largest job of the instance due to (2.1). Due to Lemma 5.1, an optimal splitting schedule $S^{*}_{split}(I_1)$ has makespan

$$C_{\max}\left(S^{*}_{split}(I_1)\right) = \frac{p(N)}{m},$$

so that

$$\frac{C_{\max}\left(S^{*}_{np}(I_1)\right)}{C_{\max}\left(S^{*}_{split}(I_1)\right)} = m\frac{p_1}{p(N)} = \lambda,$$

which satisfies (5.5). Furthermore, the maximum value for λ is achieved for an instance with a single job, i.e., $n = 1$, where $\lambda = m$.

For an instance I_2 of Class 1, where $n > m$, the algorithm finds a required schedule at Step 3 by running Algorithm *P-nonpreemptive*. Rustogi and Strusevich (2013) prove that this obtains a non-preemptive schedule $S_{np}(I_2)$, with makespan

$$C_{\max}\left(S_{np}\left(I_2\right)\right) \le \left(2 - \frac{2}{m-g+1}\right) \frac{p\left(N\right) - \sum_{j=1}^{g} p_j}{m-g}.$$

Due to Lemma 5.4, it is known that

$$\frac{C_{\max}\left(S_{np}\left(I_2\right)\right)}{C_{\max}\left(S^*_{split}\left(I_2\right)\right)} \le 2 - \frac{2}{m-g+1}$$

which satisfies (5.12). A tight instance for this class is detailed in Section 3.1, for which

$$C_{\max}\left(S_{np}\left(I_2\right)\right) = 2m - 2g$$

and

$$\begin{aligned} C_{\max}\left(S^*_{split}\right) &= \frac{p\left(N\right)}{m} \\ &= \frac{g\left(m-g+1\right) + \left(m-g+1\right)\left(m-g\right)}{m} \\ &= \frac{m\left(m-g+1\right)}{m} \\ &= m-g+1, \end{aligned}$$

so that

$$\begin{aligned} \frac{C_{\max}\left(S_{np}\left(I_2\right)\right)}{C_{\max}\left(S^*_{split}\left(I_2\right)\right)} &= \frac{2m-2g}{m-g+1} \\ &= 2 - \frac{2}{m-g+1}, \end{aligned}$$

which satisfies (5.12).

For Class 2 instances the algorithm finds a non preemptive schedule S^*_{np} at Step 4, by running algorithm *P-nonpreemptive*. Following the tight instance described for this class in Section 3.1, given an instance with $m+1$ unit length jobs, the algorithm finds a non-preemptive schedule S^*_{np} with makespan

$$C_{\max}\left(S^*_{np}\right) = 2,$$

while the makespan of the splitting schedule S^*_{split}, is given by

$$C_{\max}\left(S^*_{split}\right) = \frac{m+1}{m},$$

which yields

$$\begin{aligned}\frac{C_{\max}\left(S^*_{np}\right)}{C_{\max}\left(S^*_{split}\right)} &= \frac{2}{\frac{m+1}{m}} \\ &= 2\frac{m}{m+1} \\ &= 2 - \frac{2}{m+1},\end{aligned}$$

as required. □

5.2.2 Total Completion Time

For the problem of scheduling n jobs on m identical machines to minimize the total completion time $\Phi(S) = \sum_{j=1}^{n} C_j(S)$, let τ_m be an upper bound on the power of splitting, so that (5.3) holds across all instances of the problem. An instance I is called tight if, for that instance, the equality

$$\frac{\Phi\left(S^*_{np}\right)}{\Phi\left(S^*_{split}\right)} = \tau_m.$$

For the problem on uniform machines, Epstein et al. (2016) introduce the concept of a *tight sequence* for the power of preemption, which they define as a sequence of instances whose sequence of cost ratios approaches ρ_m, and prove that for the power of preemption there is a tight sequence in which all jobs are have equal processing requirements, which may be normalized to have unit lengths, so that $p_j = 1$, $J_j \in N$.

Using the same reasoning as that employed in Epstein et al. (2016) it is straightforward to verify that any instance of the problem of minimizing the total completion time on identical machine can be transformed into an instance in which all processing times are equal, and that transformation does not increase the power of splitting. This is due to the fact that both values $\Phi\left(S^*_{np}\right)$ and $\Phi\left(S^*_{split}\right)$ are linear functions of p_j.

Due to this observation, in order to determine a bound on the power of splitting we only need to consider the instances of the problem for which $p_j = 1$, $J_j \in N$.

Consider the problem of minimizing the total completion time on m identical machines. As shown in Lemma 5.2, every job J_j is processed on each of the machines for

p_j/m amount of time. Moreover, fraction of a job is processed within the same time interval on all of the machines, with all machines terminating at the same time.

For the single machine version of this problem, the value of the total completion time is given by

$$\Phi\left(S^*\right) = \sum_{j=1}^{n} p_j\left(n-j+1\right).$$

Due to the conditions of Lemma 5.2, for m identical parallel machines, we obtain

$$\begin{aligned}\Phi\left(S^*_{Split}\right) &= \frac{1}{m}\left(\sum_{j=1}^{n} p_j\left(n-j+1\right)\right) \\ &= \frac{1}{m}\left(\frac{1}{2}n^2+\frac{1}{2}n\right) \\ &= \frac{1}{2}n\frac{n+1}{m} \\ &= \frac{n\left(n+1\right)}{2m}.\end{aligned} \tag{5.14}$$

For a non-preemptive schedule consider the following two classes of instances based on the value of the objective function.

Class 1

For instances where there are at most as many jobs as machines, so that $n \le m$, an optimal non-preemptive schedule is obtained by assigning exactly one job to every machine, so that all jobs in the schedule start at time 0. In this case, the total completion time is given by the sum of processing times, i.e.,

$$\Phi\left(S^*_{np}\right) = p\left(N\right) \tag{5.15}$$

In the following theorem we derive the bounds for the power of splitting for instances of this class.

Theorem 5.3. *For the case $n \le m$,*

$$\frac{\Phi\left(S^*_{np}\right)}{\Phi\left(S^*_{Split}\right)} = \tau_m \le m,$$

and this bound is tight.

Proof:

From (5.14) and (5.15) we obtain

$$\begin{aligned}\frac{\Phi\left(S^*_{np}\right)}{\Phi\left(S^*_{Split}\right)} &= \frac{n}{\frac{n(n+1)}{2m}} \\ &= \frac{2m}{n+1}.\end{aligned}$$

Observe that in the above expression, the maximum is achieved for $n = 1$, so that

$$\tau_m \leq m$$

as required.

To demonstrate that this bound is tight, consider an instance with a single job, i.e., $n = 1$. It follows that $\Phi\left(S^*_{np}\right) = 1$ and $\Phi\left(S^*_{Split}\right) = 1/m$, so that $\Phi\left(S^*_{np}\right)/\Phi\left(S^*_{Split}\right) = m$. □

Class 2

Consider the set of instances where the number of jobs exceeds the number of machines, so that $n > m$. In Section 2.1.2 we showed that the sum of completion times for a non-preemptive schedule S^*_{np} is given by

$$\Phi\left(S^*_{np}\right) = \sum_{j=1}^{n} \left\lceil \frac{j}{m} \right\rceil. \tag{5.16}$$

Theorem 5.4. *For the case $n \geq m$,*

$$\frac{\Phi\left(S^*_{np}\right)}{\Phi\left(S^*_{Split}\right)} = \tau_m \leq \frac{4}{3},$$

and this bound is tight.

Proof:

We split our consideration into two cases.

Case 2.1

In this case the number of jobs is a multiple of the number of machines, so that for an integer $k \geq 1$, $n = k \cdot m$.

By substituting in (5.16), we obtain

$$\begin{aligned}\Phi\left(S^*_{np}\right) &= \sum_{i=1}^{k} mi \\ &= \frac{1}{2}km\,(k+1) \\ &= \frac{1}{2}n\,(k+1)\,.\end{aligned}$$

Thus we obtain

$$\begin{aligned}\frac{\Phi\left(S^*_{np}\right)}{\Phi\left(S^*_{Split}\right)} &= \frac{\frac{1}{2}n\,(k+1)}{\frac{1}{2}n\frac{n+1}{m}} = \\ &= m\frac{k+1}{n+1} \\ &= \frac{n+m}{n+1}.\end{aligned}$$

The maximum is achieved for $m = n$, thus

$$\tau_m \leq 2 - \frac{2}{n+1}.$$

The global bound is achieved for $m = n = 2$, which gives

$$\tau_m \leq \frac{4}{3},$$

as required.

Case 2.2

In this case the number of jobs is given by

$$n = km + r$$

where $1 \le r \le m-1$. By substituting in (5.16), we obtain

$$\begin{aligned}
\Phi\left(S^*_{np}\right) &= m\left(\sum_{i=1}^{k} mi + r\left(k+1\right)\right) \qquad (5.17)\\
&= m\left(r\left(k+1\right) + \frac{1}{2}km\left(k+1\right)\right)\\
&= \frac{1}{2}m\left(2r+km\right)\left(k+1\right)\\
&= \frac{1}{2}\left(m+km\right)\left(2r+km\right)\\
&= \frac{1}{2}\left(n+m-r\right)\left(r+n\right).
\end{aligned}$$

Observe that (5.17) achieves its maximum for

$$r = \frac{1}{2}m,$$

at which point the value of the objective function is given by

$$\Phi\left(S^*_{np}\right) = \frac{1}{8}\left(m+2n\right)^2.$$

Thus

$$\begin{aligned}
\frac{\Phi\left(S^*_{np}\right)}{\Phi\left(S^*_{Split}\right)} &= \frac{\frac{1}{8}\left(m+2n\right)^2}{\frac{1}{2}n\left(n+1\right)}\\
&= \frac{1}{4n}\frac{\left(m+2n\right)^2}{n+1}.
\end{aligned}$$

The global bound is achieved for $m=2$ and $n=3$, so that $r=\frac{1}{2}m$. This gives

$$\tau_m \le \frac{1}{4\times 3}\frac{\left(3+2\times 3\right)^2}{3+1} = \frac{4}{3}.$$

This proves the theorem. □

5.3 Power of Splitting on Uniform Machines

So far we have viewed the power of splitting, which for a given problem instance is defined by the ratio of an objective function of a non-preemptive schedule, over the objective function of a schedule with splitting jobs. In the previous section we consider this ratio for identical machines. In this section, we investigate the power of splitting for the problem of minimizing the makespan for the case where machines are uniform. Recall that in the uniform machine case, each machine M_i is associated with a particular speed s_i, so that the process time p_j of some job J_j on machine M_j is given by p_j/s_j. Throughout the following sections, we assume without loss of generality that jobs are numbered in non-increasing order of their processing requirement, according to (2.1), and machines are numbered in non-increasing order of their speeds, according to (2.15).

From Lemma 5.2, it is simple to see that the makespan of an optimal schedule with splitting jobs $C_{\max}\left(S^*_{split}\right)$ is equal to the average machine load T_m, so that

$$C_{\max}\left(S^*_{split}\right) = T_m = \frac{p\left(N\right)}{s\left(\mathcal{M}\right)}, \tag{5.18}$$

where $s\left(\mathcal{M}\right)$ is the sum of machine speeds, i.e.

$$s\left(\mathcal{M}\right) = \sum_{i=1}^{m} s_i.$$

In the following sections we determine the power of splitting for a fixed number of uniform machines.

5.3.1 Two Uniform machines

Consider initially the problem of minimizing the makespan on two uniform machines. In this problem we have machines M_1 and M_2, which for simplicity are assumed to have speeds $s_1 = s$, $s \geq 1$, and $s_2 = 1$. From (5.18), the makespan of an optimal schedule with splitting jobs, and consequently the average machine load, T_2, are given by

$$C_{\max}\left(S^*_{split}\right) = T_2 = \frac{p\left(N\right)}{s+1}.$$

Based on the value of λ, as defined in (5.4), we consider the following classes of instances for this problem.

Class 1 Instances

For instances of this class, the makespan of a preemptive schedule, $C_{\max}\left(S_p^*\right)$, is determined by the size of the largest job in the schedule, so that

$$C_{\max}\left(S_p^*\right) = \frac{p_1}{s}.$$

Hence, the value of λ for instances of this class is given by

$$\lambda = \frac{\frac{p_1}{s}}{T_2} > 1.$$

Lemma 5.5. *Given a Class 1 instance of the two uniform machine problem, for the power of splitting τ_2, it holds that*

$$\tau_2 = \lambda,$$

and this bound is tight.

Proof:

For instances of this class, an optimal preemptive schedule is obtained by assigning the largest job, J_1, to the fastest machine, M_1, while any remaining jobs are assigned to machine M_2. This corresponds to an optimal non-preemptive schedule, hence we obtain.

$$C_{\max}\left(S_{np}^*\right) = C_{\max}\left(S_p^*\right) = \frac{p_1}{s}.$$

Hence we obtain

$$\tau_2 = \frac{C_{\max}\left(S_{np}^*\right)}{C_{\max}\left(S_{split}^*\right)} = \frac{\frac{p_1}{s}}{T_2} = \lambda,$$

as required. $\square$

Class 2 Instances

For instances of Class 2, the makespan of a preemptive schedule, $C_{\max}\left(S_{np}^*\right)$, is given by the average machine load, T_2, so that

$$\lambda = \frac{T_2}{T_2} = 1.$$

For this problem we extend the approach employed by Soper and Strusevich (2014a) for analyzing the power of preemption for two machines. The algorithm which finds a non-preemptive schedule is based on the following splitting procedure. For a given set of jobs, the following procedure splits the set into two subsets, with the total processing time of these subsets being less than two thirds and less than one half of the overall processing time of the set.

Procedure Split (Soper and Strusevich (2014a))

INPUT: A set of jobs $H = \{J_1, J_2, \cdots, J_h\} \subseteq N$ with $p_j \leq \frac{2}{3}p(H)$ for $1 \leq j \leq h$

OUTPUT: A partition of set H into two subsets H' and H'' such that $p(H') \leq \frac{2}{3}p(H)$ and $p(H'') \leq \frac{1}{2}p(H)$

Step 1. If there exists a job J_k such that $p_k > \frac{1}{2}p(H)$, then define

$$H' := \{J_k\}, \ H'' = H \backslash \{J_k\}$$

and go to Step 4; otherwise, go to Step 2.

Step 2. If there is a job J_k such that $p_k > \frac{1}{3}p(H)$ then define

$$H' := H \backslash \{J_k\}, \ H'' := \{J_k\},$$

and go to Step 4; otherwise, go to Step 3.

Step 3. Scanning the jobs in the order of their numbering, determine the job J_u, $u > 1$ such that

$$\sum_{j=1}^{u-1} p_j \leq \frac{1}{3}p(H), \ \sum_{j=1}^{u-1} p_j + p_u > \frac{1}{3}p(H)$$

and define

$$U := \{J_1, \ldots, J_{u-1}, J_u\}.$$

If

$$p(U) > \frac{1}{2}p(H)$$

then define

$$H' := U, \ H'' = H \backslash U;$$

otherwise, define

$$H' := H \backslash U, \ H'' = U.$$

Step 4. Stop.

The running time of Procedure Split is $O(n)$. Recall that a non-preemptive schedule S_{np} is defined by a partition of the set of jobs, N, into two subsets N_1, N_2, where the jobs in each of these sets are assigned to machines M_1 and M_2 respectively. The following algorithm due to Soper and Strusevich (2014a) finds a non-preemptive schedule based on Procedure Split.

Algorithm Q2 (Soper and Strusevich (2014a))

INPUT: A set of jobs $N = \{J_1, J_2, \cdots, J_n\}$ to be scheduled on two uniform machines, where job 1 has the largest processing time, and $p_1 \le sT_2$

OUTPUT: A partition of set N into two subsets N_1 and N_2 that define a non-preemptive schedule

Step 1. If $s \ge 2$, define

$$N_1 := N,\ N_2 = \emptyset$$

and go to Step 3; otherwise, go to Step 2.

Step 2. Run Procedure Split1 for $H = N$ and obtain the sets H' and H''. Define

$$N_1 := H',\ N_2 = H''.$$

Step 3. Stop.

Theorem 5.5. *Algorithm Q2 finds a non-preemptive schedule S_{np}, such that the bound*

$$\tau_2 \le \phi(s) = \begin{cases} \frac{2(s+1)}{3s}, & \text{if} \quad 1 \le s \le \frac{4}{3} \\ \frac{s+1}{2}, & \text{if} \quad \frac{4}{3} \le s \le 2 \\ \frac{s+1}{s} & \text{if} \quad s \ge 2 \end{cases}, \tag{5.19}$$

holds, and this bound is tight.

Proof:

Due to the inequality $p_1 \le sT_2$, for a schedule S_{np} found in Step 1, we have that

$$C_{\max}(S_{np}) = \frac{p(N)}{s}$$

hence we obtain

$$\tau_2 \le \frac{C_{\max}(S_{np})}{C_{\max}\left(S^*_{split}\right)} = \frac{\frac{p(N)}{s}}{T_2} = \frac{s+1}{s},$$

as required.

For the schedule defined by the partition found in Step 2, we have that $C_{\max}(S_{np}) =$

$\max\{p(H')/s, p(H'')\}$. Thus we obtain

$$\frac{p(H')}{s} \leq \frac{2}{3s}p(N) = \frac{2(s+1)}{3s}T_2$$

and

$$p(H'') \leq \frac{1}{2}p(N) = \frac{s+1}{2}T_2.$$

Observe that for $s = \frac{4}{3}$, the equality

$$\frac{2(s+1)}{3s} = \frac{s+1}{2},$$

holds, which is in accordance with (5.19).

Tightness examples of this bound are equivalent to the ones described by Soper and Strusevich (2014a) in Section 3.2.2. □

5.3.2 Three Uniform Machines

Consider the problem of minimizing the makespan on three uniform machines. For this problem there are three uniform machines M_1, M_2, M_3, with speeds s_1, s_2 and s_3 respectively. From (5.18), we determine that the makespan of an optimal schedule with splitting jobs is given by

$$C_{\max}\left(S^*_{split}\right) = T_3 = \frac{p(N)}{s_1 + s_2 + s_3}.$$

Recall from Section 3.2 that an instance I is defined by a set of processing times $\mathcal{L}_n$ and a set of machine speeds $\mathcal{M}_m$, so that $I = (\mathcal{P}_n, \mathcal{M}_m)$.

Class r Instances

For r, $1 \leq r \leq m-1$, as per Definition (2.2), the makespan of the preemptive schedule for an instance I of this class is given by

$$C_{\max}\left(S^*_p(I)\right) = \max\{T_r | 1 \leq r \leq m-1\},$$

where T_r is the average machine load of the r largest jobs on the r fastest machines. Moreover, for an instance I, consider a partial instance $I_r = (\mathcal{P}_r, \mathcal{M}_r)$, such that $\mathcal{P}_r = \{p_1, \ldots, p_r\}$, and $\mathcal{M}_r = \{M_1, \ldots M_r\}$, and a partial instance $I'_r = (\mathcal{P}'_r, \mathcal{M}'_r)$ which contains all remaining jobs and machines not in instance I_r, such that $\mathcal{P}'_r =$

$\{p_{r+1}, \ldots, p_n\}$, and $\mathcal{M}'_r = \{M_{r+1}, \ldots M_m\}$.

The following algorithm finds a non-preemptive schedule S_{CMB} in two steps. Initially, using some heuristic H, two partial non-preemptive schedules $S_H(I_r)$ and $S_H(I'_r)$ are obtained for each of the partial instances. Finally the two partial schedules are combined to produce the non-preemptive schedule for instance I.

Algorithm COMBI

INPUT: A Class r, $1 \leq r \leq m-1$ instance $I = (\mathcal{L}_n, \mathcal{M}_m)$, and a heuristic algorithm H.

OUTPUT: A non-preemptive schedule $S_{CMB}(I)$.

Step 1. Given an instance $I = (\mathcal{L}_n, \mathcal{M}_m)$ of Class r, split I into two instances $(\mathcal{L}_r, \mathcal{M}_r)$ and $(\mathcal{L}'_r, \mathcal{M}'_r)$.

Step 2. Run heuristic H twice to find schedules $S_H(I_r)$ and $S_H(I'_r)$.

Step 3. Output schedule $S_{CMB}(I)$ obtained by combining the schedules $S_H(I_r)$ and $S_H(I'_r)$.

Step 4. Stop.

Lemma 5.6. *For an instance I where $n > m$, on three uniform machines M_1, M_2, M_3, with speeds $s_1 = s_2 = s$, and $s_3 = 1$, algorithm COMBI finds a non-preemptive schedule $S_{CMB}(I)$ such that*

$$\tau \leq \max\left\{\lambda, \left(1 - \frac{s}{s+1}(\lambda - 1)\right)\phi(s)\right\},$$

where $\phi(s)$ is given by (3.12) and this bound is tight.

Proof:

In Step 1 of the algorithm we obtain instances I_r, I'_r with respective average machine loads T_r, T'_r, such that

$$T_r = \frac{\sum_{j=1}^{r} p_j}{\sum_{i=1}^{r} s_i},$$

and

$$T'_r = \frac{\sum_{j=r+1}^{n} p_j}{\sum_{i=r+1}^{m} s_i}.$$

It is straightforward to check that for an instance of Class r

$$T_r \geq T'_r.$$

Moreover, by definition, we have that

$$\lambda = \frac{C_{\max}\left(S_p^*(I)\right)}{C_{\max}\left(S_{split}^*(I)\right)} = \frac{T_r}{T_m}.$$

For this proof, we are interested in the maximum value of the ratio

$$\frac{C_{\max}\left(S_{np}^*(I)\right)}{C_{\max}\left(S_{Split}^*(I)\right)} = \frac{C_{\max}\left(S_{np}^*(I)\right)}{C_{\max}\left(S_p^*(I)\right)}\frac{C_{\max}\left(S_p^*(I)\right)}{C_{\max}\left(S_{Split}^*(I)\right)} = \frac{C_{\max}\left(S_{np}^*(I)\right)}{C_{\max}\left(S_p^*(I)\right)}\lambda.$$

Observe that for this instance, the makespan of schedule $S_{CMB}(I)$ depends on which of the two schedules found in Step 2 of the algorithm achieves the maximum value, so that

$$C_{\max}\left(S_{np}^*\right) \le C_{\max}\left(S_{CMB}\right) = \max\left\{C_{\max}\left(S_H\left(I_r\right)\right), C_{\max}\left(S_H\left(I_r'\right)\right)\right\}. \tag{5.20}$$

Consider a parameter κ, $\kappa \le 1$, such that

$$\kappa = \frac{C_{\max}\left(S_p^*\left(I_r'\right)\right)}{T_m}. \tag{5.21}$$

Thus we obtain

$$\begin{aligned}
\frac{C_{\max}\left(S_{np}^*\left(I_r\right)\right)}{C_{\max}\left(S_p^*\left(I_r\right)\right)} &\le \frac{C_{\max}\left(S_H\left(I_r\right)\right)}{C_{\max}\left(S_p^*\left(I_r\right)\right)} = \frac{C_{\max}\left(S_H\left(I_r\right)\right)}{T_r};\\
\frac{C_{\max}\left(S_{np}^*\left(\mathcal{L}_r', \mathcal{M}_r'\right)\right)}{C_{\max}\left(S_p^*\left(\mathcal{L}_r, \mathcal{M}_r\right)\right)} &\le \frac{C_{\max}\left(S_H\left(\mathcal{L}_r', \mathcal{M}_r'\right)\right)}{C_{\max}\left(S_p^*\left(\mathcal{L}_r, \mathcal{M}_r\right)\right)} = \frac{C_{\max}\left(S_H\left(\mathcal{L}_r', \mathcal{M}_r'\right)\right)}{\frac{\lambda}{\kappa}C_{\max}\left(S_p^*\left(\mathcal{L}_r', \mathcal{M}_r'\right)\right)}.
\end{aligned}$$

Using these relationships we obtain

$$\begin{aligned}
\frac{C_{\max}\left(S_{np}^*(I)\right)}{C_{\max}\left(S_p^*(I)\right)} &\le \frac{C_{\max}\left(S_{CMB}(I)\right)}{T_r}\\
&= \max\left\{\frac{C_{\max}\left(S_H\left(I_r\right)\right)}{T_r}, \frac{C_{\max}\left(S_H\left(I_r'\right)\right)}{\frac{\lambda}{\kappa}C_{\max}\left(S_p^*\left(I_r'\right)\right)}\right\}.
\end{aligned}$$

For $m = 3$, and $r = 1$, we obtain

$$\frac{C_{\max}\left(S_H\left(\mathcal{L}_r, \mathcal{M}_r\right)\right)}{T_r} = \frac{T_1}{T_1} = 1$$

.

since the one job J_1 scheduled must be placed on the first machine. Hence the power of splitting must be determined by the makespan of the schedule on the remaining machines

$$\frac{C_{\max}\left(S_{CMB}\left(I\right)\right)}{T_r} \leq \frac{C_{\max}\left(S_H\left(\mathcal{L}'_r, \mathcal{M}'_r\right)\right)}{\frac{\lambda}{\kappa} C_{\max}\left(S^*_p\left(\mathcal{L}'_r, \mathcal{M}'_r\right)\right)}.$$

Now if $C_{\max}\left(S^*_p\left(\mathcal{L}'_r, \mathcal{M}'_r\right)\right) = T'_r$, then let our heuristic H use Algorithm Q2 described in Section 5.3.1 when

$$\frac{C_{\max}\left(S_H\left(\mathcal{L}'_r, \mathcal{M}'_r\right)\right)}{C_{\max}\left(S^*_p\left(\mathcal{L}'_r, \mathcal{M}'_r\right)\right)} \leq \phi\left(s\right) = \left\{ \begin{array}{cc} \frac{2(s+1)}{3s}, & \text{if } 1 \leq s \leq \frac{4}{3} \\ \frac{s+1}{2}, & \text{if } \frac{4}{3} \leq s \leq 2 \\ \frac{s+1}{s}, & \text{if } s \geq 2 \end{array} \right\}$$

and hence

$$\frac{C_{\max}\left(S^*_{np}\left(I\right)\right)}{C_{\max}\left(S^*_p\left(I\right)\right)} \leq \max\left\{1, \frac{\kappa}{\lambda}\phi\left(s\right)\right\}.$$

Thus

$$\frac{C_{\max}\left(S^*_{np}\left(I\right)\right)}{C_{\max}\left(S^*_{Split}\left(I\right)\right)} = \frac{C_{\max}\left(S^*_{np}\left(I\right)\right)}{C_{\max}\left(S^*_p\left(I\right)\right)}\lambda \leq \max\left\{\lambda, \kappa\phi\left(s\right)\right\}.$$

Using $C_{\max}\left(S^*_p\left(\mathcal{L}'_1, \mathcal{M}'_1\right)\right) = T'_1$, we can now (for $m = 3$) derive an expression for κ

$$\begin{aligned} \kappa &= \frac{C_{\max}\left(S^*_p\left(\mathcal{L}'_1, \mathcal{M}'_1\right)\right)}{C_{\max}\left(S^*_p\left(\mathcal{L}_n, \mathcal{M}_m\right)\right)} = \frac{T'_1}{T_m} \\ &= \frac{\frac{p(N)-p_1}{\sum_{i=2}^m s_i}}{T_m} = \frac{\frac{p(N)-s_1\lambda T_m}{\sum_{i=2}^m s_i}}{T_m} = \frac{\sum_{i=1}^m s_i - s_1\lambda}{\sum_{i=2}^m s_i} \\ &= 1 - \frac{s_1\left(\lambda - 1\right)}{\sum_{i=2}^m s_i} \end{aligned}$$

$$\kappa = 1 - \frac{s_1}{s_2 + s_3}\left(\lambda - 1\right).$$

Hence, for $s_1 = s_2 = s$, $s_3 = 1$

$$\frac{C_{\max}\left(S^*_{np}\left(I\right)\right)}{C_{\max}\left(S^*_{Split}\left(I\right)\right)} \leq \max\left\{\lambda, \left(1 - \frac{s}{s+1}\left(\lambda - 1\right)\right)\phi\left(s\right)\right\}.$$

Tightness examples are scaled versions of those for when $\lambda = 1$.

#	s	p_1	p_2	p_3	p_4	$C_{\max}(S_p^*)$	S_{np}^*			$C_{\max}(S_{np}^*)$	τ
							N_1	N_2	N_3		
1	$\left[1, \frac{4}{3}\right]$	$\lambda s T_3$	$\kappa \frac{s+1}{3} T_3$	$\kappa \frac{s+1}{3} T_3$	$\kappa \frac{s+1}{3} T_3$	λT_3	1	$2, 3$	4	$\kappa \frac{2(s+1)}{3s} T_3$	$\frac{\kappa}{\lambda} \frac{2(s+1)}{3s}$
$2a$	$\left[\frac{4}{3}, 2\right]$	$\lambda s T_3$	$\kappa \frac{s+1}{2} T_3$	$\kappa \frac{s+1}{2} T_3$	$-$	λT_3	1	2	3	$\kappa \frac{s+1}{2} T_3$	$\frac{\kappa}{\lambda} \frac{s+1}{2}$
$2b$	$[2, +\infty)$	$\lambda s T_3$	$\kappa \frac{s+1}{2} T_3$	$\kappa \frac{s+1}{2} T_3$	$-$	λT_3	1	$2, 3$	$\varnothing$	$\kappa \frac{s+1}{s} T_3$	$\frac{\kappa}{\lambda} \frac{s+1}{s}$

In the above instances T_3 sets the scale, i.e. it is an input parameter. We must have $\kappa = 1 - \frac{s_1}{s_2+s_3}(\lambda - 1)$ in order for the definitions to be consistent.

If $C_{\max}\left(S_p^*\left(\mathcal{L}_1', \mathcal{M}_1'\right)\right) = \frac{p_2}{s_2}$, then we know $\frac{p_2}{s_2} \leq \frac{p_1}{s_1}$, since it belongs to Class 1 (and not Class 2). Hence the makespan is dominated by machine M_1. Our heuristic schedule would place job J_2 on the second machine and the rest of the jobs on machine M_3, where they would complete before J_2 on M_2. Hence in this situation

$$\frac{C_{\max}\left(S_{np}^*\left(I\right)\right)}{C_{\max}\left(S_{Split}^*\left(I\right)\right)} = \lambda \leq \max\left\{\lambda, \kappa\phi\left(s\right)\right\}.$$

□

CHAPTER 6

Conclusion

The main body of this work is a study of the power of preemption, and the power of splitting for classical scheduling models on identical, uniform and unrelated machines. For the objectives of minimizing the makespan and sum of completion times, we provide a review of preemptive, non-preemptive and splitting job problems.

In Chapter 1 we introduce the necessary concepts and notation for this thesis, detailing the nature of the machine environments, job characteristics, and the objective functions which are investigated. We also present a brief introduction to theory of computational complexity, and introduce the underlying optimization problems and algorithm associated with the main body of our work. The scheduling problems which are considered in this thesis are introduced in Chapter 2, where the nature of each problem is discussed, and the best known results are presented. A detailed review of the existing body of work on the power of preemption is provided in Chapter 3.

Our main results are presented in Chapter 4, where we consider the power of limited preemption for several scheduling problems with a single preemption, and Chapter 5, where we consider schedules with splitting jobs. Those are listed in the following section.

6.1 Contributions

- In Section 4.1 we prove that for three identical machines, the problem of minimizing the makespan with at most one preemption, $P3\,|\#pmtn \leq 1|\,C_{\max}$, and in consequence any problem where $\#pmtn \leq m-2$, is NP-hard in the ordinary sense by demonstrating a polynomial reduction from the well-known SUBSET-SUM problem.

- For the problem of minimizing the makespan with a single preemption on uniform machines, in Section 4.2.1 we prove that the problem of minimizing the makespan on two uniform machines with a single preemption is solvable in polynomial time, and provide an algorithm which finds an optimal schedule for problem $Q2\left|\#pmtn \leq 1\right| C_{\max}$ in $O(n)$ time. As part of our solution we also give an expression for the value of the objective function of this problem.

- In Section 4.2.3 we extend the results by Jiang et al. (2014) on the power of preemption for schedules with a single preemption of uniform machines to the general case of m machines. Here we provide a polynomial time algorithm for problem $Qm\left|\#pmtn \leq 1\right| C_{\max}$, and give tight upper bounds for the power of limited preemption for this problem.

- In Section 4.2.4 we perform a parametric analysis of the power of preemption for problem $Q3\left|\#pmtn \leq 1\right| C_{\max}$, and we provide tight upper bounds for the different classes of instances.

- For the single preemption problem on two unrelated machines, in Section 4.3 we prove that problem $R2\left|\#pmtn \leq 1\right| C_{\max}$ is NP-hard in the ordinary sense via a polynomial reduction from the PARTITION problem.

- In Section 4.3.1 we give the power of preemption for a class of instances of the problem on two unrelated machines, where an optimal preemptive schedule requires only a single preemption.

- For the problems of minimizing the makespan and the sum of completion times on identical machines with splitting jobs, in Section 5.2 we perform an analysis of the power of splitting. For these problems we provide tight bounds on the power of splitting for several classes of instances.

- For the problem of minimizing the makespan on uniform machines with splitting jobs, in Section 5.3 we extend the results by Soper and Strusevich (2014a) for the case of two uniform machines, and provide the bounds for the power of splitting for different classes this problem. Moreover, for the same problem on three uniform machines, we analyze the bounds of the power of splitting for Class $r, 1 \leq r \leq m-1$ instances.

6.2 Future Work

The completion of this work has left a number of open questions which require further research. Suggestions for future work are discussed next.

- Extension of the power of limited preemption from a single preemption to a finite number of preemptions on uniform machines.
- Investigation of the power of preemption for other classes of instances for the two unrelated machine problem, and an extension of that result to m number of machines.
- Extension of the results on the power of splitting for the three uniform machine problem to other classes of instances, and a generalization of the bounds for m machines.
- Study of the power of splitting for the problem of minimizing the sum of completion times on m uniform machines.
- Study of the power of splitting for the makespan and sum of completion times objectives on unrelated machines.

REFERENCES

Arad, D., Mordechai, Y., & Shachnai, H. (2014). Tighter Bounds for Makespan Minimization on Unrelated Machines. *arXiv preprint arXiv:1405.2530.*

Atallah, M. J. (Ed.). (1998). *Algorithms and theory of computation handbook.* CRC press.

Bijsterbosch, J., & Volgenant, A. (2010). Solving the Rectangular assignment problem and applications. *Annals of Operations Research*, 181(1), 443-462.

Birkhoff, G. (1946). *Three observations on linear algebra.* Univ. Nac. Tucumán. Revista A, 5, 147-151.

Bourgeois, F., & Lassalle, J. C. (1971). An extension of the Munkres algorithm for the assignment problem to rectangular matrices. *Communications of the ACM, 14(12)*, 802-804.

Braun, O., & Schmidt, G. (2003). Parallel Processor Scheduling with Limited Number of Preemptions. *SIAM Journal on Computing*, 32(3), 671-680.

Brucker, P. (2007). *Scheduling Algorithms* (3rd edition). Berlin: Springer.

Brucker, P., Hurink, J., Jurisch, B., & Wöstmann, B. (1997). *A branch & bound algorithm for the open-shop problem.* Discrete Applied Mathematics, 76(1), 43-59.

Burkard, R. E., & Cela, E. (1999). *Linear assignment problems and extensions* (pp. 75-149). Springer US.

Chen, B. (1991). Parametric bounds for LPT scheduling on uniform processors. *Acta Mathematicae Applicatae Sinica*, 7(1), 67-73.

Chen, B. (2004). Parallel scheduling for early completion. *Handbook of scheduling: algorithms, models, and performance analysis.* CRC Press.

REFERENCES

Chen, B., Potts, C. N., & Woeginger, G. J. (1999). A review of machine scheduling: Complexity, algorithms and approximability. *In Handbook of combinatorial optimization (pp. 1493-1641).* Springer US.

Coffman Jr, E. G., & Garey, M. R. (1993). Proof of the 4/3 Conjecture for Preemptive vs. Nonpreemptive two-processor scheduling. *Journal of the ACM (JACM), 40(5),* 991-1018.

Conway, R. W., Maxwell, W. L., & Miller, L. W. (1967). *Theory of Scheduling*, Palo Alto-London.

Cook, S. A. (1971, May). The complexity of theorem-proving procedures. *In Proceedings of the third annual ACM symposium on Theory of computing* (pp. 151-158). ACM.

Correa, J. R., Skutella, M., & Verschae, J. (2012). The Power of Preemption on Unrelated Machines and Applications to Scheduling Orders. *Mathematics of Operations Research, 37(2)*, 379-398.

Crescenzi, P., & Panconesi, A. (1991). Completeness in approximation classes. *Information and Computation*, 93(2), 241-262.

Dantzig, G. B., Orden, A., & Wolfe, P. (1955). The generalized simplex method for minimizing a linear form under linear inequality restraints*. Pacific Journal of Mathematics,* 5(2), 183-195.

Davis, E., & Jaffe, J. M. (1981). Algorithms for scheduling tasks on unrelated processors. *Journal of the ACM (JACM), 28(4),* 721-736.

De, P., & Morton, T. E. (1980). Scheduling to minimize makespan on unequal parallel processors. *Decision Sciences, 11(4),* 586-602.

Dobson, G. (1984). Scheduling independent tasks on uniform processors. *SIAM Journal on Computing, 13(4)*, 705-716.

Edmonds, J., & Karp, R. M. (1972). Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2), 248-264.

Epstein, L., Levin, A., Soper, A. J., Strusevich, V.A.(2016). Power of Preemption for Minimizing Total Completion Time on Uniform Machines.

Ford Jr, L. R., & Fulkerson, D. R. (1955). A simple algorithm for finding maximal network flows and an application to the Hitchcock problem (No. RAND/P-743). RAND CORP SANTA MONICA CA.

REFERENCES

Friesen, D. K. (1987). Tighter Bounds for LPT Scheduling on Uniform Processors. *SIAM Journal on Computing, 16(3)*, 554-560.

Gairing, M., Monien, B., & Woclaw, A. (2007). A faster combinatorial approximation algorithm for scheduling unrelated parallel machines. *Theoretical Computer Science, 380(1), 87-99.*

Garey, M. R., & Johnson, D. S. (1979). *A Guide to the Theory of NP-Completeness.* San Francisco.

Gonzalez, T., & Sahni, S. (1976). Open shop scheduling to minimize finish time. *Journal of the ACM (JACM), 23(4)*, 665-679.

Gonzalez, T., Ibarra, O. H., & Sahni, S. (1977). Bounds for LPT schedules on uniform processors. *SIAM Journal on Computing, 6(1)*, 155-166.

Gonzalez, T., & Sahni, S. (1978). Preemptive scheduling of uniform processor systems. *Journal of the ACM (JACM)*, 25(1), 92-101.

Gonzalez, T., Lawler, E. L., & Sahni, S. (1990). Optimal preemptive scheduling of two unrelated processors. *ORSA Journal on Computing, 2(3)*, 219-224.

Graham, R. L. (1966). Bounds for Certain Multiprocessing Anomalies. *Bell System Technical Journal, 45(9)*, 1563-1581.

Graham, R. L. (1969). Bounds on Multiprocessing Timing anomalies. *SIAM Journal on Applied Mathematics, 17(2)*, 416-429.

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

Hariri, A. M. A., & Potts, C. N. (1991). Heuristics for Scheduling Unrelated Parallel Machines. *Computers & operations research, 18(3)*, 323-331.

Hochbaum, D. S., & Shmoys, D. B. (1987). Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM (JACM)*, 34(1), 144-162.

Horowitz, E., & Sahni, S. (1976). Exact and Approximate Algorithms for Scheduling Nonidentical Processors. *Journal of the ACM (JACM), 23(2)*, 317-327.

REFERENCES

Horvath, E. C., Lam, S., & Sethi, R. (1977). A Level Algorithm for Preemptive Scheduling. *Journal of the ACM (JACM), 24(1)*, 32-43.

Ibarra, O. H., & Kim, C. E. (1977). Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors. *Journal of the ACM* (JACM), 24(2), 280-289.

Jiang, Y., Weng, Z., & Hu, J. (2014). Algorithms with Limited Number of Preemptions for Scheduling on Parallel Machines. *Journal of Combinatorial Optimization, 27(4)*, 711-723.

Johnson, D. S., & Garey, M. R. (1979). Computers and intractability: A guide to the theory of NP-completeness. *Freeman&Co, San Francisco*, 32.

Karp, R. M. (1972). *Reducibility Among Combinatorial Problems (pp. 85-103).* Springer US.

Kellerer, H., Mansini, R., Pferschy, U., & Speranza, M. G. (2003). An efficient fully polynomial approximation scheme for the subset-sum problem. *Journal of Computer and System Sciences*, 66(2), 349-370.

Khachiyan, L. G. (1980). Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1), 53-72.

Kovács, A. (2006). Tighter Approximation Bounds for LPT Scheduling in Two Special Cases. *In Algorithms and Complexity (pp. 187-198).* Springer Berlin Heidelberg.

Kovács, A. (2010). New Approximation Bounds for LPT Scheduling. *Algorithmica, 57(2)*, 413-433.

Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2), 83-97.

Lawler, E. L., & Labetoulle, J. (1978). On Preemptive Scheduling of Unrelated Parallel Processors by Linear Programming. *Journal of the ACM (JACM), 25(4)*, 612-619.

Labetoulle, J., Lawler, E. L., Lenstra, J. K.,& Kan, A. R. (1982). Preemptive scheduling of uniform machines subject to release dates. *In Progress in combinatorial optimization.* Waterloo, Ont., 245-261

Lin, J. H., & Vitter, J. S. (1992, July). e-Approximations with minimum packing constraint violation. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing* (pp. 771-782). ACM.

Lee, C. Y., & Strusevich, V. A. (2005). Two-machine Shop Scheduling with an Uncapacitated Interstage Transporter. *IIE Transactions, 37(8)*, 725-736.

Lenstra, J.K. & Rinnooy Kan, A.H.G. (1979), Computational Complexity of Discrete Optimization Problems, In: P.L. Hammer, E.L. Johnson and B.H. Korte, Editor(s), *Annals of Discrete Mathematics*, Elsevier, Volume 4, Pages 121-140, ISSN 0167-5060, ISBN 9780444853226

Lenstra, J. K., Shmoys, D. B., & Tardos, É. (1990). Approximation Algorithms for Scheduling Unrelated Parallel Machines. *Mathematical programming, 46(1-3),* 259-271.

Liu, J. W. S., & Liu, C. L. (1974). Performance analysis of heterogeneous multiprocessor computing systems. *Computer architectures and networks* (E. Gelenbe and R. Mahl, eds.), North Holland, 331-343.

Martello, S., Soumis, F., & Toth, P. (1997). Exact and approximation algorithms for makespan minimization on unrelated parallel machines. *Discrete applied mathematics, 75(2)*, 169-188.

McNaughton, R. (1959). Scheduling with deadlines and loss functions. *Management Science, 6(1)*, 1-12.

Mokotoff, E. (1999). Scheduling to minimize the makespan on identical parallel machines: An LP-based algorithm. *Investigacion Operative, 97-107.*

Papadimitriou, C. H. (1981). *On the Complexity of Integer Programming.* Journal of the ACM (JACM), 28(4), 765-768.

Papadimitriou, C. H. (2003). *Computational complexity.* John Wiley and Sons Ltd..

Papadimitriou, C., & Yannakakis, M. (1988, January). Optimization, approximation, and complexity classes. *In Proceedings of the twentieth annual ACM symposium on Theory of computing* (pp. 229-234). ACM.

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

Potts, C. N. (1985). Analysis of a linear programming heuristic for scheduling unrelated parallel machines. *Discrete Applied Mathematics, 10(2)*, 155-164.

Potts, C. N., & Van Wassenhove, L. N. (1992). Integrating Scheduling with Batching and Lot-Sizing: A Review of Algorithms and Complexity. *Journal of the Operational Research Society*, 43(5), 395–406. http://doi.org/10.2307/2583559

REFERENCES

Queyranne, M., & Sviridenko, M. (2002). A (2+ε)-approximation algorithm for the generalized preemptive open shop problem with minsum objective. *Journal of Algorithms*, 45(2), 202-212.

Rustogi, K., & Strusevich, V. A. (2013). Parallel Machine Scheduling: Impact of Adding Extra Machines. *Operations Research, 61(5)*, 1243-1257.

Schulz, A. S., & Skutella, M. (2002). Scheduling Unrelated Machines by Randomized Rounding. SIAM J. Discret. Math., 15(4), 450–469. http://doi.org/10.1137/S0895480199357078

Shmoys, D. B., & Tardos, É. (1993). An Approximation Algorithm for the Generalized Assignment Problem. *Mathematical Programming, 62(1-3)*, 461-474.

Serafini, P. (1996). Scheduling Jobs on Several Machines with the Job Splitting property. *Operations Research,* 44(4), 617-628

Shor, N. Z. (1972). Utilization of the operation of space dilatation in the minimization of convex functions. *Cybernetics and Systems Analysis*, 6(1), 7-15.

Sitters, R. (2005). Complexity of preemptive minsum scheduling on unrelated parallel machines. *Journal of Algorithms*, 57(1), 37-48.

Sitters, R. A. (2008). Approximability of average completion time scheduling on unrelated machines. *In Algorithms-ESA 2008 (pp. 768-779).* Springer Berlin Heidelberg.

Skutella, M. (2001). Convex quadratic and semidefinite programming relaxations in scheduling. *Journal of the ACM, 48(2), 206–242.* http://doi.org/10.1145/375827.375840

Soper, A. J., & Strusevich, V. A. (2014a). Single parameter analysis of power of preemption on two and three uniform machines. *Discrete Optimization, 12*, 26-46.

Soper, A. J., & Strusevich, V. A. (2014b). Power of Preemption on Uniform Parallel Machines. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014)}, 28*, 392-402.

Von Neumann, J. (1953). *A certain zero-sum two-person game equivalent to the optimal assignment problem.* Contributions to the Theory of Games, 2, 5-12.

Williamson, D. P., Hall, L. A., Hoogeveen, J. A., Hurkens, C. A. J., Lenstra, J. K., Sevast'Janov, S. V., & Shmoys, D. B. (1997). Short shop schedules. *Operations Research, 45(2)*, 288-294.

REFERENCES

Woeginger, G. J. (2000). A comment on scheduling on uniform machines under chain-type precedence constraints. *Operations Research Letters*, 26(3), 107-109.

Xing, W., & Zhang, J. (2000). Parallel machine scheduling with splitting jobs. *Discrete Applied Mathematics*, 103(1), 259-269.