Scheduling algorithms for energy and thermal management in computer systems

Dimitrios Letsios

UNIVERSITÉ EVRY VAL D'ESSONNE

Ecole Doctorale Sciences et Ingénierie
Laboratoire IBISC - Equipe AROBAS

THÈSE

présentée et soutenue publiquement le 22 octobre 2013
pour l'obtention du grade de

Docteur de l'Université d'Evry Val d'Essonne

Discipline: Informatique

par

Dimitrios LETSIOS

Titre:

Politiques de gestion d'Énergie et de Température dans les Systèmes Informatiques

Jury

Nikhil Bansal (reviewer)
Department of Mathematics and Computer Science
Eindhoven University of Technology

Cristoph Dürr
CNRS et LIP6
University Pierre and Marie Curie

Ioannis Milis
Department of Informatics
Athens University of Economics and Business

Yves Robert
LIP
Ecole Normale Supérieure de Lyon

Maxim Sviridenko
Department of Computer Science
University of Warwick

Denis Trystram (reviewer)
LIG
Grenoble Institute of Technology

Eric Angel (co-advisor)
IBISC
University of Evry

Evripidis Bampis (advisor)
LIP6
University Pierre and Marie Curie

Résumé

La gestion de la consommation d'énergie et de la température est devenue un enjeu crucial dans les systèmes informatiques. En effet, un grand centre de données consomme autant d'électricité qu'une ville et les processeurs modernes atteignent des températures importantes dégradant ainsi leurs performances et leur fiabilité. Dans cette thèse, nous étudions différents problèmes d'ordonnancement prenant en compte la consommation d'énergie et la température des processeurs en se focalisant sur leur complexité et leur approximabilité. Pour cela, nous utilisons le modèle de Yao et al. (1995) (modèle de variation de vitesse) pour la gestion d'énergie et le modèle de Chrobak et al. (2008) pour la gestion de la température.

Abstract

Nowadays, the energy consumption and the heat dissipation of computing environments have emerged as crucial issues. Indeed, large data centers consume as much electricity as a city while modern processors attain high temperatures degrading their performance and decreasing their reliability. In this thesis, we study various energy and temperature aware scheduling problems and we focus on their complexity and approximability.

A dominant technique for saving energy is by proper scheduling of the jobs through the operating system combined with appropriate scaling of the processor's speed. This technique is referred to as *speed scaling* in the literature. The theoretical study of speed scaling was initiated by Yao, Demers and Shenker (1995) who considered the single-processor problem of scheduling preemptively a set of jobs, each one specified by an amount of work, a release date and a deadline, so as to minimize the total energy consumption. In order to measure the energy consumption of a processor, the authors considered the well-known rule according to which the processor's power consumption is $P(t) = s(t)^\alpha$ at each time t, where $s(t)$ is the processor's speed at t and $\alpha > 1$ is a machine-dependent constant (usually $\alpha \in [2, 3]$). Here, we study speed scaling problems on a single processor, on homogeneous parallel processors, on heterogeneous environments and on shop environments. In most cases, the objective is the minimization of the energy but we also address problems in which we are interested in capturing the trade-off between energy and performance.

We tackle speed scaling problems through different approaches. For non-preemptive problems, we explore the idea of transforming optimal preemptive schedules to non-preemptive ones. Moreover, we exploit the fact that some problems can be formulated as convex programs and we propose greedy algorithms that produce optimal solutions satisfying the KKT conditions which are necessary and sufficient for optimality in convex programming. In the context of convex programming and KKT conditions, we also study the design of primal-dual algorithms. Additionally, we solve speed scaling problems by formulating them as convex cost flow or minimum weighted bipartite matching problems. Finally, we elaborate on approximating energy minimization problems that can be formulated as integer configuration linear programs. We can obtain an approximate solution for such a problem by solving the fractional relaxation of an integer configuration linear program for it and applying randomized rounding.

In this thesis, we solve some new energy aware scheduling problems and we improve the best-known algorithms for some other problems. For instance, we improve the best-known approximation algorithm for the single-processor non-preemptive energy minimization problem which is strongly $\mathcal{NP}$-hard. When $\alpha = 3$, we decrease the approximation ratio from 2048 to 20. Furthermore, we propose a faster optimal combinatorial algorithm

for the preemptive migratory energy minimization problem on power-homogeneous processors, while the best-known algorithm was based on solving linear programs. Last but not least, we improve the best-known approximation algorithm for the preemptive non-migratory energy minimization problem on power-homogeneous processors for fractional values of α. Our algorithm can be applied even in the more general case where the processors are heterogeneous and, for $\alpha_{max} = 2.5$ (which is the maximum constant α among all processors), we get an improvement of the approximation ratio from 5 to 3.08.

In order to manage the thermal behavior of a computing device, we adopt the approach of Chrobak, Dürr, Hurand and Robert (2011). The main assumption is that some jobs are more CPU intensive than others and more heat is generated during their execution. So, each job is associated with a heat contribution which is the impact of the job on the processor's temperature. In this setting, we study the complexity and the approximability of multiprocessor scheduling problems where either there is a constraint on the processors' temperature and our aim is to optimize some performance metric or the temperature is the optimization goal itself.

Acknowledgements

This thesis was realized jointly in the Algorithm's group of the laboratory IBISC at the University of Evry and in the Operations Research group of the laboratory LIP6 at the University Pierre and Marie Curie. I would like to thank all the members of these groups and the staff for their hospitality.

Of course, nothing of these would have been possible without the generous financial support by

- a research grant of the French ministry of education (sur thématiques prioritaires)
- the DEFIS program TODO, ANR-09-EMER-010
- the project ALGONOW co-financed by the European Union (European Social Fund - ESF) and greek national funds (the operational program "Education and Lifelong Learning" and the program THALES)
- a PHC CAI YUANPEI France-China bilateral project
- GDR-RO of CNRS
- a grant of the Doctorate School of Sciences and Engineering of the University of Evry

I am grateful to my advisor Evripidis Bampis for his continuous support. I thank him especially for inspiring me and learning me how to think in a simple way. I also thank my advisor in the master Ioannis Milis whose guidance was essential.

Moreover, I would like to express my deep appreciation to Eric Angel, Vincent Chau, Fadi Kacem, Alexander Kononov, Evangelos Markakis, Maxim Sviridenko and Kirk Pruhs for the pleasant and enriching cooperation I had with them.

Furthermore, I want to thank Agapi Kyriakidou for bearing with me and because she kept encouraging me most of the times. Additionally, I thank my friends Konstantinos Balamotis, Angelos Balatsoukas, Katerina Kinta, Petros Kotsalas, Panagiotis Smyrnis and Georgios Zois whose presence was very significant. I also feel very lucky and pleased to be surrounded by Giorgio Lucarelli who stood like my big brother these years.

Finally and most importantly, I am grateful to my family, my father Yannis, my mother Petroula and my little brother Manthos for supporting me by all means and being present whenever I needed them.

Contents

Chapter 1

Introduction

As the technology scales, the energy consumption of computer systems becomes a major concern. This issue touches the designers and the users of almost any computing device ranging from small portable devices to large data centers. To begin with, in server farms, energy efficiency is very important because there are large costs incurred for buying energy. Moreover, part of this energy is converted into heat which increases the overall temperature of the system and this is not desirable since high temperatures affect the processors' performance and reliability. Furthermore, in battery systems, we would like to conserve energy because lower energy implies higher lifetime of the battery. The above are the principal reasons for which the energy consumption of computing devices is a crucial topic and it has become an important field of research both in academia and in industry the past years.

Another equally important subject that bothers modern computer scientists and engineers is the thermal management in computer systems. For roughly half a century, the processing speed of computing devices has been improving at high rates based on the Moore's law. It is expected that this will be no longer possible due to the large heat dissipation of modern microprocessors. High temperatures degrade the performance and reduce the lifetime of a microprocessor. Additionally, if the value of temperature becomes too high then the processor might be permanently damaged. Therefore, in order to keep satisfying the increasing demand for performance, we need to investigate ways of maintaing the temperature of computing devices as low as possible. In this direction, computer manufacturers incorporate cooling components but these components are costly. Hence, managing the processors temperature has emerged as a really hot issue recently and necessites novel approaches.

The energy consumption and thermal behavior of computing systems has always been a concern for computer scientists. Before a decade, problems concerning these issues were mainly tackled via hardware oriented solutions. The last decade, their management is also addressed at the operating system's level. Specifically, the energy expenses and the evolution of the temperature of a processor are strongly influenced by a fundamental task of the operating system known as job scheduling. The running software on a processor is divided into jobs and a job is simply part of an executed program. Traditionally, the job scheduling task consists of deciding which job is executed at each time. In order to enforce the ability of managing the energy consumption and the temperature of computing

devices, computer manufacturers have introduced an additional task for the scheduler of the operating system known as speed scaling. At each time, the scheduler of the operating system has now to decide not only the job to be run but the processor's speed as well. Speed scaling is indeed possible nowadays. For instance, speed scaling is applied to Intel processors trough the "Turbo Boost" technology while on AMD processors it is achieved with the "PowerNow" technology. The energy and temperature of a processor can be reduced by properly adjusting its speed and, in this context, we would like to design energy aware and temperature aware scheduling algorithms for the operating system which include proper job scheduling and speed scaling policies.

An efficient scheduling algorithm should satisfy the demand for performance by executing the jobs as fast as possible, but at the same time it should reduce the processor's energy consumption and maintain its temperature as low as possible. In general, energy/temperature and performance are conflicting objectives since high processors' speeds imply good performance at the price of high energy consumption and temperatures. Hence, a successful scheduling algorithm has to be constructed so as to attain a good trade-off between energy/temperature and performance.

Today, there are several types of computing environments including small desktops with a single processor and large scale data centers with several processors. Moreover, there exist special purpose processors which have been designed to execute particular types of jobs. Due to the diversity of computing environments, the principles for designing efficient scheduling algorithms, with respect to energy/temperature and performance, might not be the same for every kind of computing system. Thus, we need to focus on each type separately.

In this thesis, we study the issue of energy and thermal management in computing systems. Our principal target is the design of energy and temperature aware scheduling algorithms. In this direction, we address several scheduling problems by considering different computational environments and various optimization goals. The main contribution is the study of different algorithmic techniques which are useful in the design of efficient scheduling algorithms taking into account energy or temperature.

1.1 Energy and Thermal Models

In this section, we describe the models that we use in this thesis in order to manage the energy consumption and the temperature of a processor. The flow of the electric current and the heat dissipation of a computing device are complex phenomena and they cannot be modeled accurately. However, there exist some well-studied approximate models in the literature that offer the possibility to study the performance and the energy/temperature in an analytical way. In this thesis, we use the speed scaling model for managing the energy. For completeness, we describe some alternative models for the energy appeared in the literature, namely the power down model and the speed scaling model combined with power down, but we do not study them. As far as the temperature is concerned, there exists a continuous thermal model combined with speed scaling for the thermal management. However, the one we study is a discrete thermal model.

Speed Scaling

The *speed scaling* model was introduced by Yao, Demers and Shenker [62] and it is based on the fact that the processors speed can be varied. Consider a processor that has to execute some jobs. The processor has to execute an amount of work in order to complete each one of these jobs. We can imagine that this amount of work corresponds to a certain number of CPU cycles. Then, we define the processor's speed (or frequency) as the amount of work it executes per unit of time. Let $s(t)$ be the processor's speed at time t. The amount of work that the processor executes during an interval of time $[a, b)$ is equal to $\int_a^b s(t)dt$.

The processor consumes an amount of energy in order to execute an amount of work. We denote by $Q(t)$ the power, i.e. the instantaneous energy consumption, of the processor at time t. According to the model in [62], the power consumption of a processor is a convex function of its speed. Specifically, at any time t, we have that

$$Q(t) = s(t)^\alpha$$

where $\alpha > 1$ is a constant which depends on the technical characteristics of the processor. For instance, the processors which are constructed according to the CMOS technology are known to satisfy the cube-root rule, i.e. $\alpha \simeq 3$ (see [14]). The energy consumption of the processor during an interval of time $[a, b)$ is equal to $\int_a^b s(t)^\alpha dt$. So, if the processor operates at a constant speed s during an interval of time $[a, b)$, then it executes $(b-a) \cdot s$ units of work and it consumes $(b-a) \cdot s^\alpha$ units of energy during the same interval.

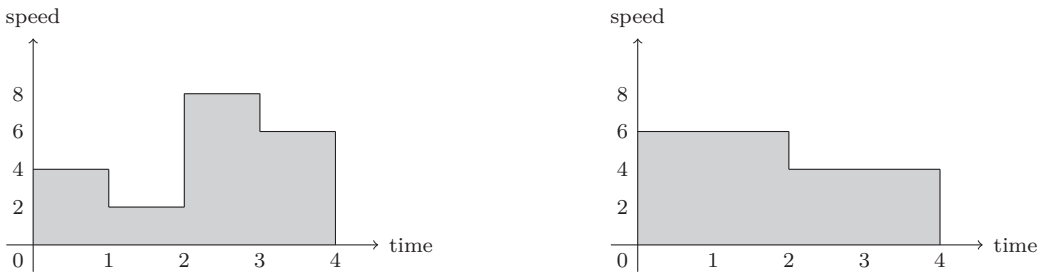


Figure 1.1: An example of two schedules for a processor whose power function is $P(t) = s(t)^2$. The processor executes $w = 20$ units of work during the interval of time $[0, 4)$ in both schedules. The first schedule consumes $E_1 = 1 \cdot 4^2 + 1 \cdot 2^2 + 1 \cdot 8^2 + 1 \cdot 6^2 = 120$ units of energy while the second one consumes $E_2 = 2 \cdot 6^2 + 2 \cdot 4^2 = 104$ units of energy.

Power Down

The *power down* model was formalized by Irani, Gupta and Shukla [45]. In this model, we assume that a processor can be in an active or in an inactive state. In the former state, we say that the processor is *ON* and that it consumes an amount of energy for being active even if nothing is executed while, in the latter state, we say that it is *OFF*,

it consumes less (or no) energy and no execution is possible. A processor can execute a job only when it is active.

For simplicity, let c be the power consumption, i.e. the energy consumption per unit of time, of the processor when it is active and assume that no energy is consumed when it is inactive. The processor can save energy by turning into the inactive state during the idle periods where there are no jobs to be executed. However, an amount of energy L is dissipated for switching back from the inactive to the active state. If a processor is active for t units of time and it performs x transitions from the inactive to the active state, then its energy consumption is

$$E = t \cdot c + x \cdot L$$

Note that, the maximum amount of time that the processor can execute any job is equal to t.

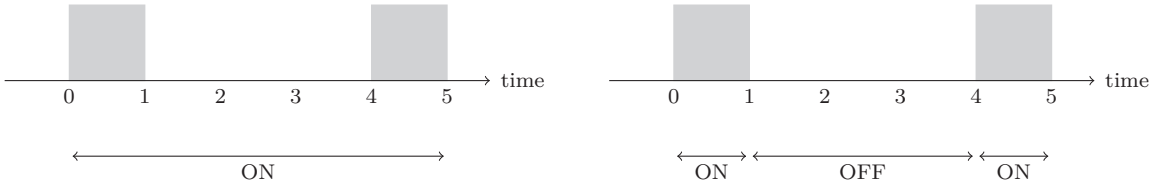


Figure 1.2: An example of two schedules for a processor which has to execute some jobs during the intervals of time $[0, 1)$ and $[4, 5)$. Its power consumption is $c = 1$ in the active state. The transition cost from the inactive to the active state is $L = 2$ units of energy. In the first schedule, the processor stays active during the whole interval $[0, 5)$ and it consumes $E_1 = 5 \cdot 1 + 0 \cdot 2 = 5$ units of energy. In the second schedule, it transitions to the inactive state at the time $t = 1$ and goes back to the active state at the time $t = 4$ and it consumes $E_2 = 2 \cdot 1 + 1 \cdot 2 = 4$ units of energy.

Power Down with Speed Scaling

There exists a hybrid model which combines speed scaling with power down mechanisms which was also introduced by Irani, Shukla and Gupta [47]. In this model, at time t, the processor's speed-to-power function is defined as

$$Q(t) = s(t)^{\alpha} + c$$

where the speed $s(t)$ and the constant α come from the standard speed scaling setting while $c > 0$ is a constant that specifies the additional power consumed at each time for being in the active state. In the inactive state, no energy is consumed. Moreover, there is an energy consumption L incurred for switching from the inactive to the active state.

Continuous Thermal Model with Speed Scaling

In the context of speed scaling, there exists a model for measuring the evolution of the processor's temperature which was introduced by Bansal, Kimbrel and Pruhs [16] and

we refer to this model as the *continuous thermal model*. According to this model, the increase of the temperature is proportional to the power supplied to the processor. Moreover, the processor's cooling is assumed to be proportional to the difference between its temperature and the ambient temperature (Newton's law of cooling). The ambient temperature Θ_0 is constant and the processor's temperature is never below Θ_0. Furthermore, the temperature scale is such that $\Theta_0 = 0$. Then, a first-order approximation for the rate of change $\Theta'(t)$ of the temperature $\Theta(t)$ at time t is

$$\Theta'(t) = b \cdot Q(t) - c \cdot \Theta(t)$$

where $Q(t)$ is the power consumption, i.e. the instantaneous energy consumption, at time t and $b, c \geq 0$ are constants. We refer to the constant c as the cooling parameter of the device. A consequence of the Newton's law of cooling is that if the processor is supplied no power, then its temperature is reduced by a constant fraction every $1/c$ units of time.

Discrete Thermal Model

The *discrete thermal model* was introduced by Chrobak, Dürr, Hurand and Robert [30]. Note that this model is not combined with speed scaling. The main assumption is that some jobs require more effort to be executed than others and, thus, more heat is generated for their execution. So, each job is associated with a heat contribution which reflects the impact of the job to the temperature when the job is executed. Moreover, the processor's cooling occurs according to the Newton's law of cooling. That is, the processor's temperature is reduced at a rate proportional to the difference of its current temperature and the ambient temperature of the processor's surroundings which is, without loss of generality, equal to zero. Furthermore, the thermal behavior of a processor depends on the technical characteristics of the processor. In order to model this, we associate each processor with constant which we call its cooling factor.

For simplicity, we assume that the time is partitioned into unit length time slots and at every such time slot either a single job is executed during the whole slot or the processor is idle. Formally, let us consider a processor with cooling factor c. Assume that, during the time slot $[t, t+1)$, the processor executes a job with heat contribution h. Let $\Theta(t)$ and $\Theta(t+1)$ be the temperatures at times t and $t+1$, respectively. Then, we have that

$$\Theta(t+1) = \frac{\Theta(t) + h}{c}$$

If the processor is idle, then the processor's temperature is modified as if a job of zero heat contribution is executed. That is, if the processor is idle during $[t, t+1)$, then

$$\Theta(t+1) = \frac{\Theta(t)}{c}$$

At this point notice that Chrobak et al. [30] studied a normalization of the discrete thermal model in which the processors have $c = 2$ and the jobs have heat contributions in the interval $[0, 2]$. In fact, this is the case we consider in this thesis.

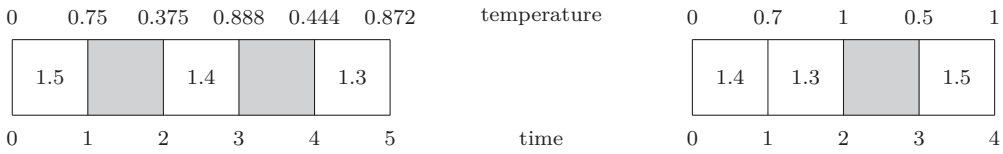


Figure 1.3: An example of two schedules for a processor whose cooling factor is $c = 2$. In both schedules, the processor executes three jobs with unit processing times and heat contributions 1.5, 1.4 and 1.3, respectively. Note that the temperature does not exceed the value 1 in any schedule.

1.2 Problem Definitions

In this section, we formally establish the setting for the scheduling problems considered in this thesis. Note that a scheduling problem is specified by a set of jobs, a processing environment and an optimization goal.

Typically, a scheduling problem consists of a set of n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$. Every job J_j has an amount of work (or processing requirement) w_j which must be executed for it. Moreover, each J_j is associated with a release date (or arrival time) r_j and a deadline d_j meaning that it can only be executed during the interval $[r_j, d_j)$. We say that J_j is *active* during $[r_j, d_j)$ and that $[r_j, d_j)$ is the *active interval* of J_j.

In general, we tackle problems such that the parameters of the jobs are arbitrary. However, we sometimes restrict our attention to special cases in which some parameters might be related. First of all, we consider problems where one of the jobs' parameters is equal for all the jobs. For example, we study a problem such that all the jobs have equal works, i.e. $w_j = w_{j'}$ for each pair of jobs $J_j, J_{j'}$, which might arise in systems that execute the same type of jobs. Furthermore, we consider problems where the active intervals of the jobs have special structures. In agreeable instances, for every couple of jobs J_j and $J_{j'}$ such that $r_j < r_{j'}$, it must be the case that $d_j \leq d_{j'}$. This kind of instances include the ones where all the jobs have active intervals of equal size and there is a sort of fairness among the jobs. We also address problems where the active intervals of the jobs have a laminar structure, that is, for every couple of jobs J_j and $J_{j'}$ such that $r_j < r_{j'}$, it holds that either $d_j \geq d_{j'}$ or $d_j \leq r_{j'}$. Note that laminar instances occur if the jobs are created by recursive calls of a program.

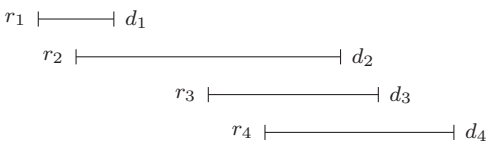


Figure 1.4: An example of an agreeable instance.

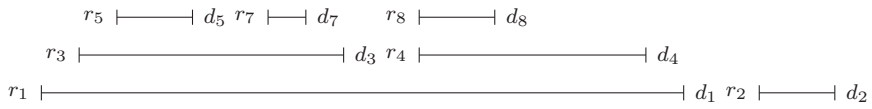


Figure 1.5: An example of a laminar instance.

In a given scheduling problem, we may or not allow preemptions and migrations of the jobs. When preemptions of jobs are permitted, a job may start its execution, be suspended and resumed later from the point of suspension. In computer systems with several processors where the jobs can be preempted, if migrations of jobs are allowed, then one job may be executed by more that one processors. However, each job can only be executed by at most one processor at each time.

For certain types of applications, there are precedence constraints among the jobs. If the job J_j is constrained to precede the job $J_{j'}$, then $J_{j'}$ cannot start its execution until J_j is completed. The precedence relations among the jobs are represented by a directed acyclic graph $G = (V, A)$. The set of vertices V of this graph contains one vertex for each job and there is an arc $(J_j, J_{j'})$ if and only if there is a constraint according to which J_j must precede $J_{j'}$.

In general, we consider different processing environments on which the jobs must be executed. In all the cases, a processor can only execute at most one job at each time. First, we consider environments with a single processor. Small portable devices are included in this type of environments. Today, for improving the performance of modern computing systems, designers use parallelism, i.e. multiple processors running at lower frequencies but offering better performances than a single processor. So, we also study multiprocessor environments consisting of a set of m processors $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$ which run in parallel and they obey to the same speed-to-power function $Q(t) = s(t)^\alpha$.

Another characteristic of the multiprocessor computing systems is that they tend to be heterogeneous consisting of processors of different types. Heterogeneity offers the possibility of further improving the performance of the system by allowing the execution of a job on the most appropriate type of processor. So, we also consider heterogeneous environments with special-purpose processors designed for particular types of jobs. In such environments, each processor P_i satisfies its own speed-to-power function $Q_i(t) = s(t)^{\alpha_i}$. Furthermore, a processor P_i may execute a job J_j more efficiently than another processor $P_{i'}$. That is, P_i might need to execute less work than $P_{i'}$ in order to complete J_j. Therefore, each job J_j is associated with a set of values $w_{i,j}$ which correspond to the amount of work that the processor P_i has to execute in order to complete J_j.

Additionally, every job J_j might have processor-dependent release dates $r_{i,j}$ and deadlines $d_{i,j}$. Scheduling problems with processor-dependent release dates and deadlines have been studied in the literature to model the situation in which the processors are connected by a network. In this case, it is assumed that every job is initially available at some processor and a transfer time must elapse before it becomes available for a new processor. The transfer time is reflected by an increase in the release date and the deadline.

In this thesis, we also consider a special type of processing environments known as shop environments. A typical shop environment consists of a set of m special-purpose parallel processors $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$. There is a set of n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ and, now, every job $J_j \in \mathcal{J}$ consists of n_j operations $O_{1,j}, O_{2,j}, \ldots, O_{n_j,j}$. Every operation $O_{k,j}$ has an amount of work $w_{k,j}$. The processors in $\mathcal{P}$ are special-purpose in the sense that each processor P_i is designed to execute a particular type of operations. Therefore, every operation $O_{k,j}$ is associated with a processor P_i on which it must be entirely executed. In a shop environment we assume that all the operations of a job access a common resource which is dedicated to that job. As a result, two operations of the same job cannot be executed simultaneously.

We consider two kinds of shop environments, namely the open shop and the job shop. In an open shop environment, each job J_j can have at most one operation on each processor. In a job shop environment, a job J_j can have more than one operations on the same processor and there are precedence constraints among the operations of each job in the form of a chain. Specifically, the operations of the job J_j are numbered as $O_{1,j}, O_{2,j}, \ldots, O_{n_j,j}$ and they must be executed in this order. That is, the operation $O_{k+1,j}$ can start only once the operation $O_{k,j}$ has finished.

Next, we elaborate on the optimization goals of the scheduling problems that we study in this thesis. Firstly, we consider the objective of minimizing the total energy consumption. Recall that our study of the energy is based on the model of Yao et al. [62] by performing speed scaling. In most of the energy-related problems studied in this thesis, there is always an optimal schedule where each job J_j is executed with a single speed s_j and this comes from the convexity of the speed-to-power function. In such schedules, we only have to define one speed for each job and the energy consumption for executing a job J_j is $E_j = w_j s_j^{\alpha-1}$. Therefore, our objective function is to minimize $E = \sum_{J_j \in \mathcal{J}} E_j = \sum_{J_j \in \mathcal{J}} w_j s_j^{\alpha-1}$. In the context of shop environments, it holds that each operation $O_{k,j}$ is executed at a constant speed $s_{k,j}$ and the total energy consumption in an optimal schedule is $\sum_{O_{k,j} \in \mathcal{O}} w_{k,j} s_{k,j}^{\alpha-1}$, where $\mathcal{O}$ is the set of all the operations.

Moreover, we study objective functions related with the thermal management. Recall that, in all the temperature aware scheduling problems that we tackle in this thesis, we adopt the discrete thermal model of Chrobak et al. [30] and, in this model, each job J_j is associated with a heat contribution h_j. We, first, have to ensure that the processors' temperature does not become too high at any time. In order to accomplish this, we consider scheduling problems where the objective is to minimize the maximum temperature Θ_{max} attained at any time, i.e. $\Theta_{max} = \max_{t \in T}\{\Theta(t)\}$, where T is the time horizon. Another objective that we address and concerns the overall thermal behavior of a computing system is the minimization of the average temperature $\sum_{t \in T} \Theta(t)$.

Finally, we consider scheduling problems where the goal is to achieve high performance under energy or thermal limitations. Specifically, we try to optimize some performance metric under either a budget of energy E or a temperature threshold Θ which must not be exceeded. In general, good performance means that the completion times of the jobs are as low as possible. We denote the completion time of the job J_j by C_j .

There exist many well-known performance metrics of a schedule in the bibliography. A first metric is the makespan C_{max} which corresponds to the time at which the last job completes, i.e. $C_{max} = \max_{J_j \in \mathcal{J}}\{C_j\}$. Clearly, we would like to construct schedules with

minimum makespan. A generalization of the makespan is the maximum lateness of a schedule. When this objective is considered, we assume that, once the job J_j has been completed, an additional amount of time $q_j \geq 0$ has to elapse until it is delivered. The parameter q_j is known as the delivery time of the job. Then, the lateness of a job J_j is defined as $L_j = C_j + q_j$ and the maximum lateness of the schedule is the maximum lateness among the jobs, i.e. $L_{max} = \max_{J_j \in \mathcal{J}}\{L_j\}$. The objective now is to minimize the maximum lateness.

Another classical metric of the quality of a schedule is the average (or total) completion time $\sum_{J_j \in \mathcal{J}} C_j$ of all the jobs. In the literature, there exists a generalization of this objective, namely the total flow time $\sum_{J_j \in \mathcal{J}} F_j$ of the jobs, where the flow time of a job is defined as $F_j = C_j - r_j$. We also consider the weighted versions of these objectives. In this case, each job J_j has a weight $\beta_j > 0$ which specifies its relevant importance with respect to the other jobs. The higher the weight, the higher the importance of the job is. A schedule with good performance should minimize either $\sum_{J_j \in \mathcal{J}} \beta_j C_j$ or $\sum_{J_j \in \mathcal{J}} \beta_j F_j$.

In energy-efficient scheduling problems, another type of objective functions is to optimize a linear combination of the energy and a performance metric. For instance, we consider problems in which we would like to minimize the energy consumption plus β times the maximum lateness, where $\beta > 0$ is a parameter specifying the relevant importance between the energy and the maximum lateness. The motivation of such a problem comes from an economic viewpoint. Specifically, we assume that we are willing to pay β units of energy in order to get a reduction of one unit of maximum lateness. So, in order to minimize our cost, it suffices to minimize the maximum lateness plus β times the energy.

1.3 Notation for Scheduling Problems

In this section, we describe a notation for energy and temperature aware scheduling problems which is a natural adaptation of the well-known three-field notation of Graham, Lawler, Lenstra and Rinnooy Kan [37] for classical scheduling problems. According to this notation, a scheduling problem is denoted by an expression with three fields in the form $f_1|f_2|f_3$. The field f_1 corresponds to the processing environment, the field f_2 concerns the jobs' characteristics and the field f_3 specifies the objective function.

In the field for the processing environment f_1, we add the parameter **S** to specify that the processors are speed-scalable or the parameter **T** for the problems under the discrete thermal model. If these terms are omitted, then we consider a classical scheduling problem without energy and thermal considerations where each job J_j has a fixed processing time. In order to indicate the processing environment, we use one of the following parameters.

1	Single Processor
P	Homogeneous Parallel Processors
R	Heterogeneous Parallel Processors
O	Open Shop
J	Job Shop

Table 1.1: Processing Environments for the 1^{st} field of the 3-field Notation

As far as the job characteristics are concerned, we use w_j (or $w_{i,j}$ in the case of heterogeneous or shop environments) for specifications on the works of the jobs (or operations). We use these parameters if we want to indicate that the jobs have equal works by writing $w_j = w$. If we omit the term, then the jobs (or the operations) have arbitrary works. In problems without speed scaling, i.e. the ones under the discrete thermal model or the ones without energy/thermal considerations, we use p_j for the processing times of the jobs instead of w_j. We write r_j and d_j (or $r_{i,j}$ and $d_{i,j}$) for clarifications concerning the release dates and the deadlines of the jobs. If the parameter r_j is not included in the 3-field notation, then all the jobs are available to the system at the time $t = 0$. Otherwise, if the jobs do not have equal release dates, then we have to add r_j. By omitting d_j, we mean that the jobs do not have deadlines. In order to indicate that the jobs have equal or arbitrary deadlines, we write $d_j = d$ and d_j, respectively. In problems under the discrete thermal model, we add the term h_j to specify that every job has a heat contribution. Note that, in the case of the maximum lateness objective, each job J_j is associated with a delivery time q_j and we do not add anything in the field f_2. By including the term *agrbl* or *lmnr*, the problem is restricted to agreeable or laminar instances, respectively. The default setting in our notation is that we do not allow preemptions and migrations of the jobs. In order to permit them, we must include the parameters *pmtn* or *mgtn*, respectively, in f_2. Finally, we add the term *prec* so as to indicate that there are precedence constraints among the jobs. The possible expressions that concern the jobs' characteristics are summarized in the following table.

$\mathbf{w_j = w}$ (or $\mathbf{w_{i,j} = w}$)	Equal-Work Jobs (Operations)
$\mathbf{p_j = p}$	Equal Processing Times
$\mathbf{r_j}$ (or $\mathbf{r_{i,j}}$)	Arbitrary Release Dates
$\mathbf{d_j}$ (or $\mathbf{d_{i,j}}$)	Arbitrary Deadlines
$\mathbf{d_j = d}$	Equal Deadlines
$\mathbf{h_j}$	Heat Contributions
agrbl	Agreeable Instances
lmnr	Laminar Instances
pmtn	Preemptions
mgtn	Preemptions and Migrations
prec	Precedence Constraints

Table 1.2: Expressions for the 2^{nd} field of the 3-field Notation

Finally, in the field f_3, we specify the objective function of the problem. In the case where the objective function is a performance-related objective function with a constraint

on the energy or the temperature, we have to indicate whether we have a budget of energy or a temperature threshold by adding in parantheses the symbols E or Θ, respectively. The possible objective functions are stated in the following table.

$\mathbf{E}$	Energy
$\mathbf{\Theta_{max}}$	Maximum Temperature
$\sum \mathbf{\Theta_t}$	Average Temperature
$\mathbf{C_{max}(E)}$ or $\mathbf{C_{max}(\Theta)}$	Makespan
$\mathbf{C_{max}} + \beta\mathbf{E}$	Makespan plus Energy
$\mathbf{L_{max}(E)}$ or $\mathbf{L_{max}(\Theta)}$	Maximum Lateness
$\mathbf{L_{max}} + \beta\mathbf{E}$	Maximum Lateness plus Energy
$\sum \mathbf{C_j(E)}$ or $\sum \mathbf{C_j(\Theta)}$	Average Completion Time
$\sum \mathbf{C_j} + \beta\mathbf{E}$	Average Completion Time plus Energy
$\sum \mathbf{w_j C_j(E)}$ or $\sum \mathbf{w_j C_j(\Theta)}$	Weighted Average Completion Time
$\sum \mathbf{w_j C_j} + \beta\mathbf{E}$	Weighted Average Completion Time plus Energy
$\sum \mathbf{F_j(E)}$ or $\sum \mathbf{F_j(\Theta)}$	Total Flow Time
$\sum \mathbf{F_j} + \beta\mathbf{E}$	Total Flow Time plus Energy
$\sum \mathbf{w_j F_j(E)}$ or $\sum \mathbf{w_j F_j(\Theta)}$	Weighted Total Flow Time
$\sum \mathbf{w_j F_j} + \beta\mathbf{E}$	Weighted Total Flow Time plus Energy

Table 1.3: Objective Functions for the 3^{rd} field of the 3-field Notation

For example, $\mathbf{S}, \mathbf{1}|\mathbf{r_j}|\mathbf{L_{max}(E)}$ is the problem of minimizing the maximum lateness of a set of jobs with release dates where the objective is the minimization of the maximum lateness and there is a budget of energy. In the problem $\mathbf{S}, \mathbf{R}|\mathbf{r_{i,j}}, \mathbf{d_{i,j}}, \mathbf{mgtn}|\mathbf{E}$, we would like to minimize the energy of a set of jobs with processor-dependent release dates and deadlines on fully heterogeneous parallel processors where preemptions and migrations of jobs are allowed. Finally, in $\mathbf{T}, \mathbf{P}|\mathbf{p_j} = \mathbf{1}, \mathbf{d_j} = \mathbf{d}, \mathbf{h_j}|\mathbf{\Theta_{max}}$, our objective is to minimize the maximum temperature on parallel identical processors under the discrete thermal model, where there is a set of unit-length jobs with equal release dates and deadlines.

1.4 Algorithm Analysis

Tractability and Approximation Algorithms

The *running time* of an algorithm is the number of elementary operations it performs such as primitive arithmetic operations, primitive logic operations etc. A *polynomial algorithm* for a given optimization problem, is an algorithm which produces an optimal solution for the problem in time polynomial the size of its instance $|I|$, i.e. the number of the bits needed in order to encode the instance I in a binary representation.

We say that an optimization problem is *tractable* if it admits a polynomial algorithm. In general, there exist problems which are tractable and others which are intractable. However, there is also a class of problems, the *$\mathcal{NP}$-complete problems*, for which we do not know whether they are tractable or not. A basic aspect of the $\mathcal{NP}$-complete problems is that they all have, in a sense, equivalent difficulty. Specifically, if there was a tractable

$\mathcal{NP}$-complete problem, then this would imply tractability for every other $\mathcal{NP}$-complete problem. On the other hand, if there was an intractable $\mathcal{NP}$-complete problem, then this would imply intractability for every other $\mathcal{NP}$-complete problem. The question of the tractability of $\mathcal{NP}$-complete problems in a major open question in computer science and it is known as the $\mathcal{P} = \mathcal{NP}$ question. In general, it is conjectured that $\mathcal{P} \neq \mathcal{NP}$ which means that the $\mathcal{NP}$-complete problems are intractable. The opposite is considered unlikely.

The *equivalence* property of the $\mathcal{NP}$-complete problems, provide a way of showing that a problem is $\mathcal{NP}$-complete through a so called $\mathcal{NP}$*-completeness reduction*. Specifically, assume that we are given an optimization problem Π and that we know that another problem Π' is $\mathcal{NP}$-complete. In order to show that Π is $\mathcal{NP}$-complete, it suffices to show that if we are given an optimal polynomial algorithm for Π', then we can use it as a black box an define an optimal algorithm for Π.

Unless $\mathcal{P} = \mathcal{NP}$, we do not expect a polynomial-time algorithm for an $\mathcal{NP}$-complete problem. However, many $\mathcal{NP}$-complete problems are very important in practice and we would like to cope with them. One way to solve such a problem is by an *approximation algorithm*. An approximation algorithm is a polynomial-time algorithm which does not produce an optimal solution but a near-optimal solution instead. Formally, consider an optimization problem for which we are given a polynomial-time algorithm A. For a given instance I of the problem, we denote by $C_A(I)$ and $C_{OPT}(I)$ the cost of the algorithm's solution and the cost of the optimal solution, respectively. Then, A is a ρ-approximation algorithm if, for any possible instance I of the problem, it holds that

$$C_A(I) \leq \rho \cdot C_{OPT}(I)$$

If a problem admits a ρ-approximation algorithm, then we can compute, in polynomial time, a solution whose cost is at most ρ times the cost of an optimal solution. We refer to the value ρ as the *approximation ratio* of the algorithm A. For some $\mathcal{NP}$-complete problems, we may define a *polynomial time approximation scheme* (PTAS) which is an algorithm that computes a solution whose cost is very close to the optimal. Formally, a PTAS is an algorithm which computes an $(1+\epsilon)$-approximate solution in time polynomial to the size of the instance, for any $\epsilon > 0$. When an algorithm computes an $(1 + \epsilon)$-approximate solution in time polynomial to the size of the instance and $1/\epsilon$, for any $\epsilon > 0$, then we call it a *fully polynomial time approximation scheme* (FPTAS).

Online Algorithms

Our discussion so far has lied around the *offline* setting. That is, we assume that the algorithm knows the entire instance before solving a problem. This is not the case in the *online* setting in which the algorithm does not know all the instance in advance but the knowledge comes over the time while the algorithm runs. In order to evaluate the performance of online algorithms for some optimization problem, we adopt the *competitive analysis* according to which the solution of an algorithm is compared with the solution of an optimal offline algorithm. Assume that we are given an online algorithm A for some optimization problem. For a given instance I of the problem, let $C_A(I)$ and $C_{OPT}(I)$ the cost of an algorithm's solution and the cost of the optimal offline solution, respectively.

We say that A is ρ-competitive if, for any possible instance I of the problem, it holds that

$$C_A(I) \leq \rho \cdot C_{OPT}(I)$$

1.5 Related Work

In this section, we will describe existing work on energy and temperature aware scheduling problems which is closely related to this thesis. Initially, we present part of the literature for speed scaling problems on a single processor, on homogeneous parallel processors and on heterogeneous parallel processors. We also briefly describe existing work on the power down model and the hybrid model that combines speed scaling with power down. Note that there exist some surveys in the context of energy-efficient scheduling by Albers [2] and by Irani and Pruhs [46]. Finally, we present existing work for thermal management problems.

Speed Scaling on a Single Processor

Offline Energy Minimization. The theoretical study of speed scaling was initiated in a seminal paper by Yao et al. [62] who considered the single processor problem of scheduling a set of jobs with release dates and deadlines, preemptively, so as to minimize the total energy consumption, i.e. $\mathbf{S}, \mathbf{1}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{pmtn}|\mathbf{E}$. The authors showed that the particular problem is polynomially solvable by constructing an optimal algorithm whose running time is $O(n^3)$. Later, Li et al. [52] proposed a faster algorithm with time complexity $O(n^2 \log n)$. When the instances are restricted to be laminar, Li et al. [51] showed that the problem can be solved in $O(n)$ time.

Antoniadis et al. [9] were the first to consider the non-preemptive energy minimization problem $\mathbf{S}, \mathbf{1}|\mathbf{r_j}, \mathbf{d_j}|\mathbf{E}$ for which they observed that it is strongly $\mathcal{NP}$-hard even for laminar instances. They also presented a $2^{4\alpha-3}$-approximation algorithm for laminar instances and a $2^{5\alpha-4}$-approximation algorithm for general instances. Furthermore, the authors noticed that the problem can be solved optimally in polynomial time when the instances are agreeable by observing that the optimal preemptive schedule produced by the algorithm in [62] is always non-preemptive.

Problem	Complexity	Best-known Algorithm
$\mathbf{S,1\vert r_j,d_j,pmtn,lmnr\vert E}$	Polynomial	$O(n)$ [51]
$\mathbf{S,1\vert r_j,d_j,pmtn\vert E}$	Polynomial	$O(n^2 \log n)$ [52]
$\mathbf{S,1\vert r_j,d_j,agrbl\vert E}$	Polynomial	$O(n^3)$ [9] [62]
$\mathbf{S,1\vert r_j,d_j,lmnr\vert E}$	$\mathcal{NP}$-hard	$2^{4\alpha-3}$-approximation [9]
$\mathbf{S,1\vert r_j,d_j\vert E}$	$\mathcal{NP}$-hard	$2^{5\alpha-4}$-approximation [9]

Table 1.4: Offline Energy Minimization

Online Energy Minimization. Yao et al. [62] considered also the online version of the problem $\mathbf{S,1\vert r_j,d_j,pmtn\vert E}$ in which each job is known at its release date. They proposed two reasonable online algorithms, namely the AVR (Average Rate) and the OA (Optimal Available). For AVR, they established a competitive ratio of $2^{\alpha-1}\alpha^\alpha$ and they showed that it cannot be better than α^α. Later, Bansal et al. [12] presented a more elementary (simpler) proof of the fact that AVR is $2^{\alpha-1}\alpha^\alpha$-competitive and they concluded that this ratio is almost tight by showing that AVR's competitive ratio cannot be less than $(2-\delta)^{\alpha-1}\alpha^\alpha$, where δ approaches zero as α goes to infinity. In another work, Bansal et al. [16] proved that OA is α^α-competitive and they showed that this ratio is essentially tight for OA, because there is an instance such that the energy consumption of the OA's schedule is α^α times the energy consumption of an optimal offline schedule. In the same work, they proposed the BKP algorithm with competitive ratio $2(\frac{\alpha}{\alpha-1})^\alpha e^\alpha$, which is better than OA for $\alpha \geq 5$. Finally, Bansal et al. [14] defined the qOA algorithm which is $\frac{4^\alpha}{2e^{1/2}\alpha^{1/4}}$-competitive. Moreover, the authors showed that qOA cannot be better than $\frac{4^{\alpha-1}}{\alpha}(1-\frac{2}{\alpha})^{\alpha/2}$-competitive and they established a generic lower bound $\frac{e^{\alpha-1}}{\alpha}$ on the competitive ratio of any deterministic algorithm for the problem.

Algorithm	Competitive Ratio	
	Lower Bound	Upper Bound
AVR	$(2-\delta)^{\alpha-1}\alpha^\alpha$ [12]	$2^{\alpha-1}\alpha^\alpha$ [62]
OA	α^α [16]	α^α [16]
BKP		$2(\frac{\alpha}{\alpha-1})^\alpha e^\alpha$ [16]
qOA	$\frac{4^{\alpha-1}}{\alpha}(1-\frac{2}{\alpha})^{\alpha/2}$ [14]	$\frac{4^\alpha}{2e^{1/2}\alpha^{1/4}}$ [14]
Any Deterministic	$\frac{e^{\alpha-1}}{\alpha}$ [14]	

Table 1.5: Online Energy Minimization

Next, we consider single processor speed scaling problems where the objective is a performance criterion under a budget of energy.

Offline Average Completion Time Minimization. The first work in this context was by Pruhs et al. [56] who considered the problem of minimizing the average completion time under a budget of energy and proposed an $O(n^2 \log \frac{E}{\epsilon})$ polynomial time algorithm for the special case where the jobs have equal works, where E is the energy budget and ϵ

is the desired accuracy. In another work, Albers et al. [5] proposed a simplified algorithm for the problem of minimizing the average completion time plus energy which is based on dynamic programming. These results hold for the objective of minimizing the total flow time under a budget of energy as well.

Megow et al. [54] considered the weighted version of the average completion time objective. For the case where all the jobs have equal release dates, they established a polynomial time approximation scheme (PTAS) and, interestingly, they showed that this problem is equivalent to the problem $\mathbf{1}||\sum \mathbf{w_j}(\mathbf{C_j})^{\frac{\alpha-1}{\alpha}}$ in which no speed scaling is performed and every job has a fixed processing time. The complexity status of the latter problem is an open question. Independently from [54], the equivalence of $\mathbf{1}||\sum \mathbf{w_j}\mathbf{C_j}(\mathbf{E})$ with $\mathbf{1}||\sum \mathbf{w_j}(\mathbf{C_j})^{\frac{\alpha-1}{\alpha}}$ was also shown by Vásquez [60]. For the preemptive problem $\mathbf{S},\mathbf{1}|\mathbf{r_j},\mathbf{pmtn}|\sum \mathbf{w_j}\mathbf{C_j}(\mathbf{E})$, where the jobs have arbitrary release dates, Megow et al. [54] proposed a $(2+\epsilon)$-approximation algorithm.

Problem	Complexity	Best-known Algorithm
$\mathbf{S},\mathbf{1}\|\mathbf{r_j},\mathbf{p_j}=\mathbf{p}\|\sum \mathbf{C_j}(\mathbf{E})$	Polynomial	$O(n^2 \log \frac{E}{\epsilon})$ [56]
$\mathbf{S},\mathbf{1}\|\|\sum \mathbf{w_j}\mathbf{C_j}(\mathbf{E})$	?	PTAS [54]
$\mathbf{S},\mathbf{1}\|\mathbf{r_j},\mathbf{pmtn}\|\sum \mathbf{w_j}\mathbf{C_j}(\mathbf{E})$	$\mathcal{NP}$-hard	$(2+\epsilon)$-approximation [54]

Table 1.6: Offline Average Completion Time Minimization

Online Total Flow Time. For the online version of the average completion time minimization problem with a budget of energy $\mathbf{S},\mathbf{1}|\mathbf{r_j},\mathbf{pmtn}|\sum \mathbf{C_j}(\mathbf{E})$, where each job is known only once it has arrived (i.e. at its release date), it is not possible to have a constant factor competitive algorithm even if we consider instances with unit-work jobs. A formal proof of this invariant was presented by Bansal et al. [17] where they proposed an adversarial strategy which makes any deterministic algorithm run out of energy. For this reason, in order to optimize both the average completion time and the energy in the online setting, Albers et al. [5] proposed to study problems where the objective function is the sum of the two objectives.

Albers et al. [5] initiated the study of the online non-preemptive energy-efficient problem $\mathbf{S},\mathbf{1}|\mathbf{r_j}|\sum \mathbf{F_j}+\mathbf{E}$ for which they showed that the best possible algorithm cannot be better than $\Omega(n^{1-1/\alpha})$-competitive. So, they considered the case where the jobs have unit works and they proposed an $O(1)$-competitive algorithm whose competitive ratio is $8(1+\Phi)^{\alpha}(\frac{\alpha}{\alpha-1})^{\alpha}$. Next, Bansal et al. [17] improved this result by showing that the algorithm in [5] is 4-competitive for unit-work jobs. Since an optimal preemptive schedule is non-preemptive for unit-work jobs, the competitive ratio of the algorithm in [5] is the same for the preemptive case as well.

Bansal et al. [17] studied the more general problem $\mathbf{S},\mathbf{1}|\mathbf{r_j},\mathbf{pmtn}|\sum \mathbf{F_j}+\mathbf{E}$ where the jobs have arbitrary release dates and preemptions are allowed. They constructed an algorithm with a competitive ratio equal to $(1+\epsilon)\max\{2, \frac{2(\alpha-1)}{\alpha-(\alpha-1)^{1-\frac{1}{\alpha-1}}}\}$. When the value of α is large, this ratio is approximately $2(\frac{\alpha}{\ln \alpha})^2$. Later, Lam et al. [49] proposed a better algorithm of competitive ratio $\frac{2}{1-(\alpha-1)/(\alpha^{\frac{\alpha}{\alpha-1}})}$. This ratio tends to $2\frac{\alpha}{\ln \alpha}$ for large values

of α. Next, Bansal et al. [15] made significant progress on this problem by presenting a 3-competitive algorithm. Finally, Andrew et al. [7] established the best online algorithm for the problem which is a slight modification of the one in [15] and which is 2-competitive. Moreover, they showed a lower bound of 2 on the competitive ratio of any algorithm in a class of reasonable algorithms.

As far as the online problem $\mathbf{S, 1|r_j, pmtn|\sum w_j F_j + E}$ of minimizing the weighted flow time is concerned, no deterministic algorithm can be $O(1)$-competitive and this holds even for the classical scheduling setting where no speed scaling is performed. The proof of this negative result was due to Bansal et al. [13].

Problem	Lower Bound	Best-known Algorithm
$\mathbf{S, 1\|r_j\|\sum F_j + E}$	$\Omega(n^{1-\frac{1}{\alpha}})$ [5]	
$\mathbf{S, 1\|r_j, pmtn\|\sum F_j + E}$		2-competitive [7]
$\mathbf{S, 1\|r_j, pmtn\|\sum w_j F_j + E}$	no $O(1)$-competitive [13]	

Table 1.7: Online Total Flow Time Minimization

Offline Makespan Minimization. Bunde [28] studied the non-preemptive offline problem $\mathbf{S, 1|r_j|C_{max}(E)}$ of minimizing the makespan of a set of jobs with release dates under an energy budget. Specifically, he proposed an optimal polynomial-time algorithm with running time $O(n^2)$. Note that, for the preemptive case of the problem, there is always an optimal schedule which is non-preemptive. Therefore, the algorithm in [28] is optimal for the preemptive case, too.

Speed Scaling on Homogeneous Parallel Processors

Offline Energy Minimization. Chen et al. [29] were the first to study a multiprocessor energy-efficient scheduling problem involving speed scaling. More specifically, they proposed a polynomial-time algorithm for solving optimally the multiprocessor migratory preemptive energy minimization problem of a set of jobs with equal release dates and deadlines. The running time of their algorithm is $O(n \log n)$. Later, Bingham et al. [23] constructed an optimal algorithm for the general version of the problem $\mathbf{S, P|r_j, d_j, mgtn|E}$ where the jobs have arbitrary release dates and deadlines. The algorithm in [23] makes repetitive calls of a black-box algorithm for solving linear programs. Then, Albers et al. [4] presented a faster combinatorial algorithm which is based on a formulation of the problem as a maximum flow problem. It has to be noticed here that, independently, we presented another optimal polynomial time algorithm for the same problem which is based on the relation of the problem with the maximum flow problem.

Albers et al. [6], considered the non-migratory preemptive problem of minimizing the energy of a set of unit-work jobs with arbitrary release dates and deadlines. The authors showed that this problem can be solved optimally in polynomial time if the instances are restricted to be agreeable. Moreover, they established an $\mathcal{NP}$-hardness proof for the unit-work case when the release dates and the deadlines of the jobs are arbitrary and they proposed an $\alpha^\alpha 2^{4\alpha}$-approximation algorithm for it. They also produced an

algorithm of the same approximation ratio for arbitrary-work instances when the jobs have either equal release dates or equal deadlines. Next, Greiner et al. [39] presented a $B_{\lceil\alpha\rceil}$-approximation algorithm for the general problem $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{pmtn}|\mathbf{E}$ with jobs having arbitrary processing requirements, where $B_{\lceil\alpha\rceil}$ is the $\lceil\alpha\rceil$-th Bell number.

Very little attention has been given to the non-migratory non-preemptive problem $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}|\mathbf{E}$. Albers et al. [6] observed that the problem is $\mathcal{NP}$-hard even in the special case where the jobs have the same release date and the same deadline. Moreover, they claimed that, for this special case of the problem, there exists a polynomial time approximation scheme (PTAS) which can be derived easily from an existing PTAS of the well-known problem $\mathbf{P}||\mathbf{C_{max}}$.

Problem	Complexity	Best-known Algorithm
$\mathbf{S}, \mathbf{P}|\mathbf{d_j} = \mathbf{d}, \mathbf{mgtn}|\mathbf{E}$	Polynomial	$O(n \log n)$ [29]
$\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$	Polynomial	max-flow based [4]
$\mathbf{S}, \mathbf{P}|\mathbf{w_j} = \mathbf{1}, \mathbf{r_j}, \mathbf{d_j}, \mathbf{agrbl}, \mathbf{pmtn}|\mathbf{E}$	Polynomial	$O(mn^2 \log n)$ [6]
$\mathbf{S}, \mathbf{P}|\mathbf{w_j} = \mathbf{1}, \mathbf{r_j}, \mathbf{d_j}, \mathbf{pmtn}|\mathbf{E}$	$\mathcal{NP}$-hard	$\min\{\alpha^\alpha 2^{4\alpha}, B_{\lceil\alpha\rceil}\}$-approximation [6] [39]
$\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{pmtn}|\mathbf{E}$	$\mathcal{NP}$-hard	$B_{\lceil\alpha\rceil}$-approximation [39]
$\mathbf{S}, \mathbf{P}|\mathbf{d_j} = \mathbf{d}|\mathbf{E}$	$\mathcal{NP}$-hard	PTAS [6]

Table 1.8: Offline Energy Minimization

Online Energy Minimization. For the online version of $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$, Albers et al. [4] proposed the online algorithms AVR and OA which are the straightforward generalizations of the corresponding algorithms for the single processor case presented in [62]. In [4], they showed that AVR is $\frac{(3\alpha)^\alpha}{2} + 2^\alpha$-competitive and that OA is α^α-competitive.

Albers et al. [6] considered the online variant of the problem $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{pmtn}|\mathbf{E}$ and restricted their attention to unit work instances. For agreeable instances, they constructed a $2(\frac{\alpha}{\alpha-1})^\alpha e^\alpha$-competitive algorithm while, for the case where the release dates and the deadlines of the jobs are arbitrary, they developed an $\alpha^\alpha 2^{4\alpha}$-competitive algorithm. For general instances with arbitrary release dates and works, Bell et al. [21] proposed an online algorithm with competitive ratio $2^{4\alpha}(\log^\alpha P + \alpha^\alpha 2^{\alpha-1})$, where P is the ratio of the maximum work among the jobs over the minimum work.

Problem	Best-known Algorithm
$\mathbf{S}, \mathbf{P}|\mathbf{p_j} = \mathbf{1}, \mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$	α^α-competitive [4]
$\mathbf{S}, \mathbf{P}|\mathbf{p_j} = \mathbf{1}, \mathbf{r_j}, \mathbf{d_j}, \mathbf{agrbl}, \mathbf{pmtn}|\mathbf{E}$	$2(\frac{\alpha}{\alpha-1})^\alpha e^\alpha$-competitive [6]
$\mathbf{S}, \mathbf{P}|\mathbf{p_j} = \mathbf{1}, \mathbf{r_j}, \mathbf{d_j}, \mathbf{pmtn}|\mathbf{E}$	$\alpha^\alpha 2^{4\alpha}$-competitive [6]
$\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{pmtn}|\mathbf{E}$	$2^{4\alpha}(\log^\alpha P + \alpha^\alpha 2^{\alpha-1})$-competitive [21]

Table 1.9: Online Energy Minimization

Online Total Flow Time Minimization. For the problem $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{pmtn}|\sum \mathbf{F_j} + \mathbf{E}$, Lam et al. [48], proposed an online algorithm whose competitive ratio is $O(2\alpha(\log P + 2^{\alpha}))$. Moreover, on the negative side, Leonardi et al. [50] showed that no deterministic algorithm can be $O(1)$-competitive even for processors with fixed speeds which is extended to the speed scaling setting.

Offline Makespan Minimization. Pruhs et al. [57] studied the non-migratory multiprocessor problem $\mathbf{S}, \mathbf{P}||\mathbf{C_{max}}(\mathbf{E})$ of minimizing the makespan of a set of jobs with equal release dates under a budget of energy and derived a PTAS for it by using as a black box an existing PTAS for the classical scheduling problem of minimizing the ℓ_α norm of a load balancing problem. Moreover, they considered the more general version where there are precedence constraints among the jobs and they proposed an $O(\log^{1+\frac{2}{\alpha}} m)$-approximation algorithm for it.

Speed Scaling on Heterogeneous Parallel Processors

There does not exist much work on environments with heterogeneous processors. In [41] and [42], Gupta et al. considered the online problem of minimizing the flow time plus energy and they presented online algorithms with a constant competitive ratio which are based on resource augmentation. These works indicate that energy efficient scheduling on heterogeneous processors may be more difficult than the homogeneous case and new algorithms are required.

Power Down

The power down model was formalized by Irani, Gupta and Shukla [45]. Baptiste [18] considered the single processor problem of minimizing the energy of a set of jobs with release dates and deadlines. He proposed an optimal algorithm for jobs with unit processing times a FPTAS for the more general case where the jobs have arbitrary processing times and preemptions are allowed. Later, Baptiste et al. [20] proposed a faster polynomial algorithm for unit jobs and presented a polynomial algorithm for the preemptive problem with arbitrary processing times. Further results with respect to this model can be found in [8], [31] and [32].

Power Down with Speed Scaling

The model that combines speed scaling with power down was first studied by Irani et al. [47] who derived a constant factor approximation for the problem of minimizing the energy of a set of jobs with release dates and deadlines. Then, Albers et al. [3] showed that the problem is $\mathcal{NP}$-hard if the power function is of a particular form. They also proposed an improved approximation algorithm. Finally, Bampis et al. [11] proved that the problem is polynomially solvable for agreeable instances.

Continuous Thermal Model

The continuous thermal model was introduced by Bansal et al. [16]. First, they considered the offline problem $\mathbf{S}, \mathbf{1}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{pmtn}|\mathbf{\Theta_{max}}$ of minimizing the maximum temperature and they showed that it can be solved in polynomial time by applying the Ellipsoid Algorithm. In the same work, they proposed an $e^{\alpha}2^{\alpha+1}(6(\frac{\alpha}{\alpha-1})^{\alpha}+1)$-competitive algorithm for the online version of the problem of minimizing the maximum temperature in which each job is known at its release date. Atkins et al. [10] developed a faster $O(n^2)$ combinatorial algorithm for the offline case where the jobs have equal release dates. Moreover, they defined another algorithm for the online case with arbitrary release dates whose competitive ratio is $\frac{e}{e-1}(2+3e\alpha^{\alpha})$. This algorithm is better than the one in [16] for some values of α, e.g. when the cube-root rule $\alpha = 3$ holds.

Discrete Thermal Model

The study of temperature-aware scheduling problems with respect to the discrete thermal model was initiated by Chrobak et al. [30] who considered the single-processor problem of finding schedules with maximum throughput for unit jobs. They showed that the problem is strongly $\mathcal{NP}$-hard even when the jobs have equal release dates and deadlines and unit processing times and the processor's cooling factor is $c = 2$. In this problem it is possible that we cannot schedule feasibly all the jobs between their release dates and their deadlines and our objective is to maximize the number of jobs which are completed on time.

The $\mathcal{NP}$-hardness proof in [30] implies that the problems $\mathbf{T}, \mathbf{1}|\mathbf{p_j} = \mathbf{1}, \mathbf{d_j} = \mathbf{d}, \mathbf{h_j}|\mathbf{\Theta_{max}}$ (maximum temperature minimization), $\mathbf{T}, \mathbf{1}|\mathbf{p_j} = \mathbf{1}, \mathbf{h_j}|\sum \mathbf{F_j}(\mathbf{\Theta})$ (total flow time minimization) and $\mathbf{T}, \mathbf{1}|\mathbf{p_j} = \mathbf{1}, \mathbf{h_j}|\mathbf{C_{max}}(\mathbf{\Theta})$ (makespan minimization) are also $\mathcal{NP}$-hard.

For the problem of minimizing the total flow time, Birks et al. [27] proposed a 2.618-approximation algorithm for the special case where all the jobs are released at the same time and they established an $\Omega(n^{1/2-\epsilon})$-inapproximability result for instances with arbitrary release dates, where $\epsilon > 0$.

Chrobak et al. [30] also considered the online problem of maximizing the throughput in which the jobs arrive over time and they proposed an algorithm with constant competitive ratio. Then, Birks et al. [24], [25], [26] addressed several generalizations of the online throughput maximization problem. In fact, in [24] the weighted throughput objective is considered. In [25] the cooling effect is generalized by multiplying the temperature by 1/c, where c > 1, instead of one half, while in [26] the jobs have equal (non-unit) processing times. Finally, Dürr et al. [34] considered the offline problem of maximizing the throughput and proposed positive and negative results on the approximation ratio of the coolest first algorithm.

1.6 Contributions

In this section, we briefly describe the contributions of this thesis.

Single Processor

Initially, we consider the single-processor non-preemptive energy minimization problem $\mathbf{S}, \mathbf{1}|\mathbf{r_j}, \mathbf{d_j}|\mathbf{E}$. Recall that the study of this problem was initiated recently and it was observed that the problem is strongly $\mathcal{NP}$-hard [9].

Antoniadis et al. [9] proposed a constant factor approximation algorithm for the non-preemptive problem through a transformation to the unrelated machine scheduling problem with the ℓ_α-norm objective. Here, we explore the idea of transforming an optimal preemptive schedule to a non-preemptive one and we show that, for unit-work instances, this approach leads to an improved approximation ratio. In Section 2.1, we derive some properties of optimal preemptive schedules produced by the algorithm of Yao et al. [62]. Next, in Section 2.2 we prove that the preemptive optimal solution does not preserve enough of the structure of the non-preemptive optimal solution and, more precisely, that the ratio between the energy consumption of an optimal non-preemptive schedule and the energy consumption of an optimal preemptive one can be $\Omega(n^{\alpha-1})$. So, with this approach, we obtain an $(1 + \frac{w_{max}}{w_{min}})^\alpha$-approximation algorithm, where $\frac{w_{max}}{w_{min}}$ is the ratio between the maximum and the minimum work among the jobs. For equal-work instances, this algorithm is 2^α-approximate which is better than the $2^{5\alpha-4}$-approximation algorithm by Antoniadis et al. [9] proposed for arbitrary work instances.

Next, we follow another approach for solving $\mathbf{S}, \mathbf{1}|\mathbf{r_j}, \mathbf{d_j}|\mathbf{E}$ which based on a reduction of the problem to the multiprocessor non-migratory preemptive energy minimization problem $\mathbf{S}, \mathbf{P}|\mathbf{r_{i,j}}, \mathbf{d_{i,j}}, \mathbf{pmtn}|\mathbf{E}$ in which the release dates and the deadlines of the jobs are processor-dependent. Our reduction allows us to prove that based on a ρ-approximation algorithm for the latter problem, we obtain a $2^{\alpha-1}\rho$-approximate solution for the former one.

In Section 2.3, we initiate the study of the single-processor scheduling problem of minimizing the maximum lateness and the energy of a set of jobs. Initially, we address the problem of minimizing the maximum lateness under a budget of energy and we propose an optimal polynomial-time algorithm for the special case in which the jobs have equal release dates, i.e. for the problem $\mathbf{S}, \mathbf{1}||\mathbf{L_{max}}(\mathbf{E})$. This algorithm constructs greedily an optimal solution satisfying the KKT conditions applied to a convex programming formulation of the problem. Subsequently, we show that the problem $\mathbf{S}, \mathbf{1}|\mathbf{r_j}|\mathbf{L_{max}}(\mathbf{E})$ in which the jobs may have arbitrary release dates is strongly $\mathcal{NP}$-hard. Finally, we move our attention to the online setting in which each job is known at its release date. Clearly, given the existing literature (see Bansal et al. [17]), we do not expect a constant factor competitive algorithm for the problem of minimizing the maximum lateness under a budget of energy. For this reason, following the approach of Albers et al. [5] for the average completion time objective, we study the online problem $\mathbf{S}, \mathbf{1}|\mathbf{r_j}|\mathbf{L_{max}} + \beta\mathbf{E}$ of minimizing a linear combination of the maximum lateness and the energy and we obtain a 2-competitive algorithm by applying a batched scheduling strategy [59].

Homogeneous Parallel Processors

Subsequently, we study multiprocessor scheduling problems on homogeneous parallel processors. Initially, we address the multiprocessor problem of minimizing the energy of a set of jobs on parallel homogeneous processors where preemptions and migrations of jobs

are allowed, i.e. $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$. Recall that the previously best known algorithm for this problem uses an optimal algorithm for solving linear programs as a black box. So, in Section 3.1, we present a faster combinatorial algorithm which is based on maximum flow computations. Note that, independently from the algorithm presented in this thesis, another algorithm was proposed for the same problem by Albers et al. [4] which also explores the relation of the problem with the maximum flow problem. These results introduce the use of maximum flow formulations in the context of speed scaling.

In order to establish the optimality of our maximum flow based algorithm and of the one of Albers et al. [4] for $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$, we need a series of technical lemmas. We present an alternative algorithm for $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$ which is based on a formulation of the problem as a minimum convex cost flow problem. This algorithm constructs an optimal schedule through a single convex cost flow computation and its analysis is much simpler.

Finally, in Section 3.2, we initiate the study of the multiprocessor energy minimization problem in which migrations and preemptions of the jobs are not allowed. As for the single processor case, we study the idea of transforming optimal migratory preemptive schedules to non-preemptive ones. While for general instances we do not hope to obtain a constant-factor approximation algorithm by using this idea, we obtain a constant-factor approximation algorithm for agreeable instances. We propose an algorithm which starts by computing an optimal multiprocessor migratory preemptive schedule for the problem $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$. In this way, it calculates a processing time for each job. By speeding up the execution of each job, it constructs a feasible non-preemptive schedule and we obtain a $(2-\frac{1}{m})^{\alpha}$-approximation algorithm for agreeable instances, i.e. $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{agrbl}|\mathbf{E}$.

Heterogeneous Parallel Processors

Next, we study scheduling problems on heterogeneous processors in which each processor satisfies its own speed-to-power function and the jobs have processor-dependent processing requirements. As Gupta et al. [41] noticed, scheduling problems with heterogeneous processors seem to require new techniques in order to be solved than their corresponding with homogeneous processors. In order to solve energy minimization problems in such environments, we introduce the idea of solving and rounding configuration linear programs.

First we consider the energy minimization problem $\mathbf{S}, \mathbf{R}|\mathbf{w_{i,j}}, \mathbf{r_{i,j}}, \mathbf{d_{i,j}}, \mathbf{mgtn}|\mathbf{E}$, where preemptions and migrations of the jobs are allowed. In order to obtain an algorithm for this problem, we formulate the problem as a configuration linear program (LP) with an exponential number of variables. This configuration LP cannot be solved directly in polynomial time. However, we show how to apply the Ellipsoid algorithm to its dual LP and, then, solve the configuration linear program with only a polynomial number variables. In this way, we obtain an $(OPT+\epsilon)$-approximate solution in time polynomial to the size of the problem's instance and $1/\epsilon$.

Next, we move our attention to the problem $\mathbf{S}, \mathbf{R}|\mathbf{w_{i,j}}, \mathbf{r_{i,j}}, \mathbf{d_{i,j}}, \mathbf{pmtn}|\mathbf{E}$ in which preemptions of jobs are allowed but migrations are not permitted. This problem can be formulated as an integer configuration LP. In order to solve this LP, we show how to solve its fractional relaxation in polynomial time by applying the Ellipsoid algorithm.

Then we trasform the optimal fractional solution to a feasible integral one by applying randomized rounding. Our algorithm is $\tilde{B}_\alpha(1+\epsilon)$-approximate, where $\tilde{B}_\alpha$ is the generalized Bell number. Subsequently, we show that the algorithm can be made faster by solving a more compact LP and then transforming the optimal solution obtained into an optimal fractional solution of the configuration LP.

Shop Environments

Another type of computing environments considered in this thesis are the so called shop environments. Initially, we study the energy minimization problem $\mathbf{S},\mathbf{O}|\mathbf{d_j}=\mathbf{d},\mathbf{pmtn}|\mathbf{E}$ in an open shop environment, where preemptions of the operations are allowed. For this problem we follow two different approaches.

Firstly, we derive an optimal algorithm for $\mathbf{S},\mathbf{O}|\mathbf{d_j}=\mathbf{d},\mathbf{pmtn}|\mathbf{E}$ based on a primal-dual schema in the setting of convex programming and KKT conditions. Note that there exists much work on the use of the primal-dual method in the field of combinatorial optimization. However, most of this work concerns applications of the method in linear programming. This method was applied only recently to the more general setting of convex programming by Devanur et al. [33] and Vegh [61]. Because of the KKT conditions, the dual variables are related with the primal ones through a set of equalities. So, we obtain an optimal primal solution by properly adjusting the dual variables. We prove that our algorithm converges to the optimal solution but we are unable to prove that it converges in polynomial time. Therefore, we performed a series of experiments showing that the number of iterations of our algorithm increases linearly with the number of jobs n when it holds that $m \neq n$, where m is the number of the processors. However, in the very specific case where $n = m$, our algorithm is slower. We are also interested in the comparison of the execution time of our method with respect to the time spent by a commercial solver which solves directly the corresponding convex program.

Our second approach for solving $\mathbf{S},\mathbf{O}|\mathbf{d_j}=\mathbf{d},\mathbf{pmtn}|\mathbf{E}$ is to formulate of the problem as a minimum convex cost flow problem. The main technical difficulty behind our algorithm is that it is not obvious how the amount of flow F, which is a parameter for the minimum convex cost formulation, can be computed. However, we show a way for computing F through several minimum convex cost computations.

Next, we present a $\tilde{B}_{\alpha_{max}}$-approximation algorithm for the energy minimization problem $\mathbf{S},\mathbf{J}|\mathbf{w_{i,j}},\mathbf{r_{i,j}},\mathbf{d_{i,j}},\mathbf{pmtn}|\mathbf{E}$ in a job shop environment. This algorithm is based on solving the fractional relaxation of an integer configuration LP and applying randomized rounding in order to obtain a feasible integral solution.

Temperature Aware Scheduling

Finally, we consider scheduling problems in which our focus is no longer the management of the energy but the management of the temperature. In this thesis, we adopt the discrete thermal model introduced by Chrobak et al. [30] and we initiate the study of several multiprocessor scheduling problems such that either there is a temperature threshold which must not be exceeded, or the temperature is the optimization goal itself.

Firstly, we address the problem $\mathbf{T},\mathbf{P}|\mathbf{p_j}=\mathbf{1},\mathbf{h_j}|\mathbf{C_{max}}(\mathbf{\Theta})$ of minimizing the makespan under a temperature threshold and we solve it by transforming any instance of the

problem to an instance of the classical makespan minimization problem $\mathbf{P}||\mathbf{C_{max}}$ in which there are no thermal considerations. Then, by using any ρ-approximation algorithm for $\mathbf{P}||\mathbf{C_{max}}$ as a black box, we obtain a 2ρ-approximation algorithm for the temperature-aware problem. Given that there exists a polynomial time approximation scheme (PTAS) for $\mathbf{P}||\mathbf{C_{max}}$, our transformation leads to a $(2+\epsilon)$-approximation ratio for $\mathbf{T},\mathbf{P}|\mathbf{p_j}=\mathbf{1},\mathbf{h_j}|\mathbf{C_{max}}(\mathbf{\Theta})$ within a running time that is polynomial in n and exponential in $1/\epsilon$. If instead of the PTAS we use the standard LPT rule which is $(\frac{4}{3}-\frac{1}{3m})$-approximate for $\mathbf{P}||\mathbf{C_{max}}$, we present a tighter analysis, improving the 2ρ-approximation ratio to $(\frac{7}{3}-\frac{1}{3m})$, while the overall running time is $O(n\log n)$.

Subsequently, we study the problem $\mathbf{T},\mathbf{P}|\mathbf{p_j}=\mathbf{1},\mathbf{d_j}=\mathbf{d},\mathbf{h_j}|\mathbf{\Theta_{max}}$ of minimizing the maximum temperature and we propose a $\frac{4}{3}$-approximation algorithm. Moreover, we show that our algorithm cannot not have better approximation ratio and our analysis is essentially tight. Then, we move our attention to the problem $\mathbf{T},\mathbf{P}|\mathbf{p_j}=\mathbf{1},\mathbf{d_j}=\mathbf{d},\mathbf{h_j}|\sum\mathbf{\Theta_t}$ of minimizing the average temperature and we show that it is polynomially solvable.

The results of this thesis come from the following publications:

- E. Bampis, A. Kononov, D. Letsios, G. Lucarelli and M. Sviridenko. Energy Efficient Scheduling and Routing via Randomized Rounding. Submitted.
- E. Bampis, V. Chau, D. Letsios, G. Lucarelli and I. Milis. Energy Minimization via a Primal-dual Algorithm for a Convex Program. E. Bampis, V. Chau, D. Letsios, G. Lucarelli and I. Milis. 12th International Symposium on Experimental Algorithms (SEA'13), Rome, Italy, p. 366-377, LNCS 7933, Springer, 2013.
- E. Bampis, A. Kononov, D. Letsios and G. Lucarelli and I. Nemparis. From Preemptive to Non-preemptive Speed-Scaling Scheduling. 19th International Computing and Combinatorics Conference (COCOON'13), Hangzhou, China, p. 134-146, LNCS 7936, Springer, 2013.
- E. Bampis, D. Letsios and G. Lucarelli. Green Scheduling, Flows and Matchings. 23rd International Symposium on Algorithms and Computation (ISAAC'12), Taipei, Taiwan, p. 106-115, LNCS 7676, Springer, 2012.
- E. Angel, E. Bampis, F. Kacem and D. Letsios. Speed Scaling on Parallel Processors with Migration. 18th International European Conference on Parallel and Distributed Computing (EURO-PAR'12), Rhodes, Greece, p. 128-140, LNCS 7484, Springer, 2012.
- E. Bampis, D. Letsios, I. Milis and G. Zois. Speed Scaling for Maximum Lateness. 18th International Computing and Combinatorics Conference (COCOON'12), Sydney, Australia, p. 25-36, LNCS 7434, Springer, 2012.
- E. Bampis, D. Letsios, G. Lucarelli, E. Markakis and I. Milis. On Multiprocessor Temperature-Aware Scheduling Problems. Joint Conference of 6th International Frontiers of Algorithmics Workshop and 8th International Conference on Algorithmic Aspects of Information and Management (FAW-AAIM'12), Beijing, China, p. 149-160, LNCS 7285, Springer, 2012.

Chapter 2

Single Processor

In this chapter, we begin the study of energy efficient scheduling problems on the basic setting of a single processor.

First, in Section 2.1, we present an optimal algorithm for the preemptive energy minimization problem $\mathbf{S}, \mathbf{1}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{pmtn}|\mathbf{E}$ which was proposed by Yao et al. [62]. Then, in Section 2.2, based on this algorithm, we derive an $(1 + \frac{w_{max}}{w_{min}})^{\alpha}$-approximation algorithm for the non-preemptive energy minimization problem $\mathbf{S}, \mathbf{1}|\mathbf{r_j}, \mathbf{d_j}|\mathbf{E}$, where w_{max} and w_{min} are the maximum and the minimum work of a job, respectively. Note that our algorithm is 2^{α}-approximate for instances in which the jobs have equal works. In Section 2.2, we also propose another approximation algorithm for the non-preemptive problem $\mathbf{S}, \mathbf{1}|\mathbf{r_j}, \mathbf{d_j}|\mathbf{E}$ which based on a transformation of the problem to the multiprocessor energy minimization problem $\mathbf{S}, \mathbf{P}|\mathbf{r_{i,j}}, \mathbf{d_{i,j}}, \mathbf{pmtn}|\mathbf{E}$ where preemptions of jobs are allowed but migrations are forbidden. Given a ρ-approximation algorithm for the latter problem as a black box, we obtain a $2^{\alpha-1}\rho$-approximation algorithm for the former problem.

Subsequently, in Section 2.3, we consider offline and online energy aware problems where the objective is the minimization of the maximum lateness. Initially, we consider the offline problem $\mathbf{S}, \mathbf{1}|\mathbf{r_j}|\mathbf{L_{max}}(\mathbf{E})$ of minimizing the maximum lateness under a budget of energy. For the special case in which the jobs have equal release dates, i.e. $\mathbf{S}, \mathbf{1}||\mathbf{L_{max}}(\mathbf{E})$, we propose an optimal polynomial time algorithm. Then, we show that the problem becomes strongly $\mathcal{NP}$-hard when the release dates of the jobs may be arbitrary. Finally, we move our attention to the online problem $\mathbf{S}, \mathbf{1}|\mathbf{r_j}|\mathbf{L_{max}} + \beta\mathbf{E}$ of minimizing a linear combination of the maximum lateness and the energy. In the online setting, each job is known when it is released. For this problem, we propose a 2-competitive algorithm which schedules the jobs in batches by applying repeatedly an optimal offline algorithm for $\mathbf{S}, \mathbf{1}||\mathbf{L_{max}} + \mathbf{E}$. Such an algorithm for $\mathbf{S}, \mathbf{1}||\mathbf{L_{max}} + \mathbf{E}$ can be obtained by using the optimal offline algorithm for $\mathbf{S}, \mathbf{1}||\mathbf{L_{max}}(\mathbf{E})$ as a black box and applying binary search.

2.1 Energy Minimization with Preemptions

In this section, we describe an optimal algorithm for the preemptive energy minimization problem $\mathbf{S}, \mathbf{1}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{pmtn}|\mathbf{E}$. This algorithm was proposed by Yao et al. [62]. Moreover, we establish some properties of the schedules produced by this algorithm that we use in Section 2.2 in order to derive an approximation algorithm for the non-preemptive problem

$\mathbf{S}, \mathbf{1}|\mathbf{r_j}, \mathbf{d_j}|\mathbf{E}$.

An instance of the problem $\mathbf{S}, \mathbf{1}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{pmtn}|\mathbf{E}$ consists of a set of jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ which have to be scheduled by a single processor. Each job $J_j \in \mathcal{J}$ has an amount of work w_j, a release date r_j and a deadline d_j. Preemptions of jobs are allowed. That is, a job may be executed, suspended and resumed later from the point of suspension. The goal is to find a minimum energy schedule such that, for each job $J_j \in \mathcal{J}$, w_j units of work are executed during the interval $[r_j, d_j)$.

We consider the time points $t_0, t_1, \ldots, t_\tau$, in increasing order, where each t_k, $0 \leq k \leq \tau$, corresponds to either a release date or a deadline, so that for each release date and deadline of a job there is a corresponding time point t_k. Then, we define the intervals $I_{k,\ell} = [t_k, t_\ell)$, for all $0 \leq k < \ell \leq \tau$, and we denote by $|I_{k,\ell}|$ the length of $I_{k,\ell}$, that is $|I_{k,\ell}| = t_\ell - t_k$. We say that a job J_j is *strictly active* in a given interval $I_{k,\ell}$, if $[r_j, d_j) \subseteq I_{k,\ell}$. The set of strictly active jobs in the interval $I_{k,\ell}$ is denoted by $\mathcal{A}(I_{k,\ell})$. The density $\delta(I_{k,\ell})$ of an interval $I_{k,\ell}$ is the total work of the jobs which are strictly active during this interval over its length, i.e. $\delta(I_{k,\ell}) = \frac{1}{|I_{k,\ell}|} \sum_{J_j \in \mathcal{A}(I_{k,\ell})} w_j$.

Yao et al. [62] proposed a polynomial-time algorithm for finding an optimal schedule for $\mathbf{S}, \mathbf{1}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{pmtn}|\mathbf{E}$. Note that there is always an optimal schedule for this problem such that each job $J_j \in \mathcal{J}$ is executed with a constant speed s_j; this is a consequence of the convexity of the speed-to-power function. This algorithm schedules the jobs in distinct phases. More specifically, in each phase, the algorithm searches for the interval $I_{k,\ell}$, $0 \leq k < \ell \leq \tau$, of the highest density. All jobs in $\mathcal{A}(I_{k,\ell})$ are assigned the same speed, which is equal to the density $\delta(I_{k,\ell})$ of the interval, and they are scheduled in $I_{k,\ell}$ using the Earliest Deadline First (EDF) policy. That is, at each time, the algorithm schedules the job with the earliest deadline. Without loss of generality, we can assume that, in the case where two jobs have the same deadline, the algorithm schedules first the job of the smallest index. Then, the set of jobs $\mathcal{A}(I_{k,\ell})$ and the interval $I_{k,\ell}$ are eliminated from the instance, the algorithm searches for the next interval of the highest density and so on. Off course, in the new critical interval, the algorithm does not take into account the subintervals in which it has already scheduled some jobs. A high-level description of the algorithm is given in Algorithm 2.1. The Figure 2.1 illustrates an example of the Algorithm 2.1.

Algorithm 2.1

1: **while** there are remaining to jobs to be scheduled **do**
2: Identify the densest critical interval $I_{k,\ell}$.
3: Schedule the remaining jobs in $\mathcal{A}(I_{k,\ell})$ with speed $\delta(I_{k,\ell})$ according to EDF, breaking ties in smallest job index first.
4: Remove these jobs and the intervals occupied by them.

Given a schedule $\mathcal{S}$ and a job J_j, let $B_j(\mathcal{S})$ and $C_j(\mathcal{S})$ be the beginning and the completion time, respectively, of J_j in $\mathcal{S}$. For simplicity, we will use B_j and C_j, if the corresponding schedule is clear from the context. Note that there are no jobs with the same beginning times, and hence all B_j's are distinct. For the same reason, all C_j's are distinct.

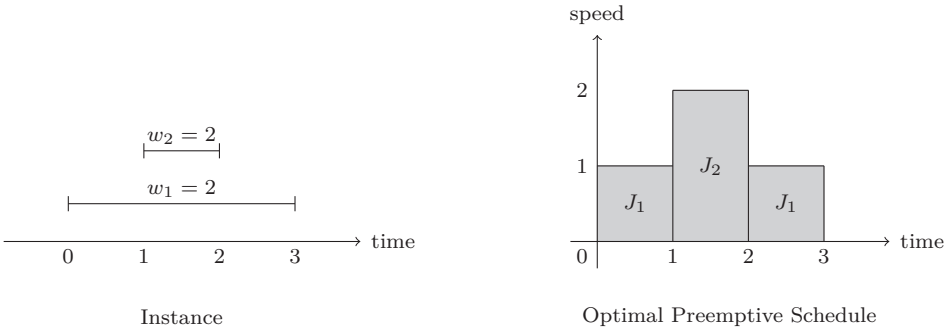


Figure 2.1: An instance with two jobs and their optimal preemptive schedule produced by the Algorithm 2.1. Initially, the densest critical interval is $[1, 2)$ and J_2 is critical job. In the second step, the densest critical interval is $[0, 3)$ and the critical job is J_1. Note that the density of the interval $[0, 3)$ is $\delta_{0,3} = 1$ in the second step of the algorithm because the job J_2 and the interval $[0, 1)$ has been removed in the first step.

The following lemma describes some structural properties of the optimal preemptive schedule created by the Algorithm 2.1.

Lemma 2.1. *Consider the optimal preemptive schedule $\mathcal{S}^*_{pr}$ created by the Algorithm 2.1. For any two jobs J_j and $J_{j'}$ in $\mathcal{S}^*_{pr}$, the following hold.*

(i) If $B_j < B_{j'}$, then either $C_j > C_{j'}$ or $C_j < B_{j'}$.

(ii) If $B_j < B_{j'}$ and $C_j > C_{j'}$, then the job J_j is not executed during the interval $(B_{j'}, C_{j'})$ and $s_j \leq s_{j'}$.

Proof.
(i) Assume for contradiction that there are two jobs J_j and $J_{j'}$ in $\mathcal{S}^*_{pr}$ with $B_j < B_{j'}$, $C_j < C_{j'}$ and $C_j > B_{j'}$.

We prove, first, that J_j and $J_{j'}$ cannot be scheduled in a different phase of the Algorithm 2.1. Without loss of generality, assume for contradiction that J_j is scheduled in a phase before $J_{j'}$ and that $I_{k,\ell}$ is the interval of the highest density in this phase. As $B_j < B_{j'} < C_j$, there is a non-empty subinterval $I \subseteq [B_{j'}, C_j] \subset [B_j, C_j] \subseteq I_{k,\ell}$ during which $J_{j'}$ is executed in $\mathcal{S}^*_{pr}$. By the definition of the Algorithm 2.1, every job is scheduled in a single phase. Moreover, the jobs scheduled at any time during $I_{k,\ell}$ cannot be scheduled after the phase at which J_j is scheduled because the interval $I_{k,\ell}$ is ignored in subsequent steps and we have a contradiction. Hence, J_j and $J_{j'}$ are scheduled in the same phase.

The Algorithm 2.1 schedules J_j and $J_{j'}$ using the EDF policy. Since the EDF policy schedules $J_{j'}$ at time $B_{j'}$ and $B_j < B_{j'} < C_j$, it holds that $d_{j'} \leq d_j$. In a similar way, since the EDF policy schedules J_j at time C_j and $B_{j'} < C_j < C_{j'}$, it holds that $d_j \leq d_{j'}$. Hence, $d_j = d_{j'}$. However, since J_j and $J_{j'}$ are available for execution and not completed at $B_{j'}$ and C_j, the algorithm should have selected the same job for execution in both

times, i.e. the job of the smallest index. Therefore, there is a contradiction on the way that Algorithm 3.1 works.

(ii) The fact that J_j cannot be scheduled during $(B_{j'}, C_{j'})$ can be proved along with the same lines with the proof of the previous item. Similarly, we can show that $J_{j'}$ cannot be scheduled at a phase after the one of J_j because no job is scheduled during $[B_j, C_j]$ once J_j has been scheduled. Hence, $s_j \leq s_{j'}$. □

The Lemma 2.1 implies that, given an optimal preemptive schedule $\mathcal{S}^*_{pr}$ for a set of jobs $\mathcal{J}$ constructed by the Algorithm 2.1, we can construct a tree representation of $\mathcal{S}^*_{pr}$. This tree representation is a directed graph $T = (V, A)$, where V is the set of vertices, A is the set of edges, and it is constructed as follows. For each job J_j we create a vertex. For each pair of jobs J_j and $J_{j'}$ with $[B_{j'}, C_{j'}] \subset [B_j, C_j]$, we create an arc $(J_j, J_{j'})$ if and only if there is not a job $J_{j''}$ with $[B_{j'}, C_{j'}] \subset [B_{j''}, C_{j''}] \subset [B_j, C_j]$. Note that the created graph $T = (V, A)$ is, in general, a forest. Moreover, by Lemma 2.1, we have that for each arc $(J_j, J_{j'})$ it holds that $s_j \leq s_{j'}$ in $\mathcal{S}^*_{pr}$. In other words, the speed of a job is at most equal to the speed of its children in T. In what follows, we denote by T_j the subtree of T rooted at the vertex $J_j \in V$. Moreover, let n_j be the number of children of J_j in T.

Lemma 2.2. *Consider an optimal preemptive schedule $\mathcal{S}^*_{pr}$ created by the Algorithm 2.1 and its tree representation $T = (V, A)$. Then, each job $J_j \in \mathcal{J}$ is preempted at most n_j times in $\mathcal{S}^*_{pr}$, where n_j is the number of children of the node J_j in T.*

Proof. Consider any job J_j. The Lemma 2.1 implies that if some job $J_{j'}$ is executed during $[B_j, C_j]$, then it must be the case that $[B_{j'}, C_{j'}] \subseteq [B_j, C_j]$. Additionally, because of Lemma 2.1, the job J_j can only be preempted at the beginning time $B_{j'}$ of a job $J_{j'}$. Clearly, there does not exist any job $J_{j''}$ such that $[B_{j'}, C_{j'}] \subseteq [B_{j''}, C_{j''}] \subseteq [B_j, C_j]$. These mean that J_j can only be preempted by its children in T and at most one time by each child. The lemma follows. □

2.2 Energy Minimization without Preemptions

In this section, we turn our attention to the non-preemptive energy minimization problem $\mathbf{S}, \mathbf{1}|\mathbf{r_j}, \mathbf{d_j}|\mathbf{E}$. First, we show that the ratio between the energy consumption of any optimal non-preemptive schedule and the energy consumption of an optimal preemptive schedule can be $\Omega(n^{\alpha-1})$. Next, we propose an $(1 + \frac{w_{max}}{w_{min}})^\alpha$-approximation algorithm which is based on the idea of transforming an optimal preemptive schedule to a non-preemptive one, where $w_{max} = \max_{J_j \in \mathcal{J}}\{w_j\}$ and $w_{min} = \min_{J_j \in \mathcal{J}}\{w_j\}$. This algorithm is 2^α-approximate for equal-work instances. Finally, we present another $2^{\alpha-1}\rho$-approximation algorithm for $\mathbf{S}, \mathbf{1}|\mathbf{r_j}, \mathbf{d_j}|\mathbf{E}$ which uses as a black box any ρ-approximation algorithm for the multiprocessor non-migratory preemptive energy minimization problem $\mathbf{S}, \mathbf{P}|\mathbf{r_{i,j}}, \mathbf{d_{i,j}}, \mathbf{pmtn}|\mathbf{E}$.

In the problem $\mathbf{S}, \mathbf{1}|\mathbf{r_j}, \mathbf{d_j}|\mathbf{E}$, there is a set of jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ which have to be executed non-preemptively on a single processor. The fact that we do not allow preemptions means that each job must be executed consecutively without any interruptions between its starting time and its completion time. Each job $J_j \in \mathcal{J}$ comes with an

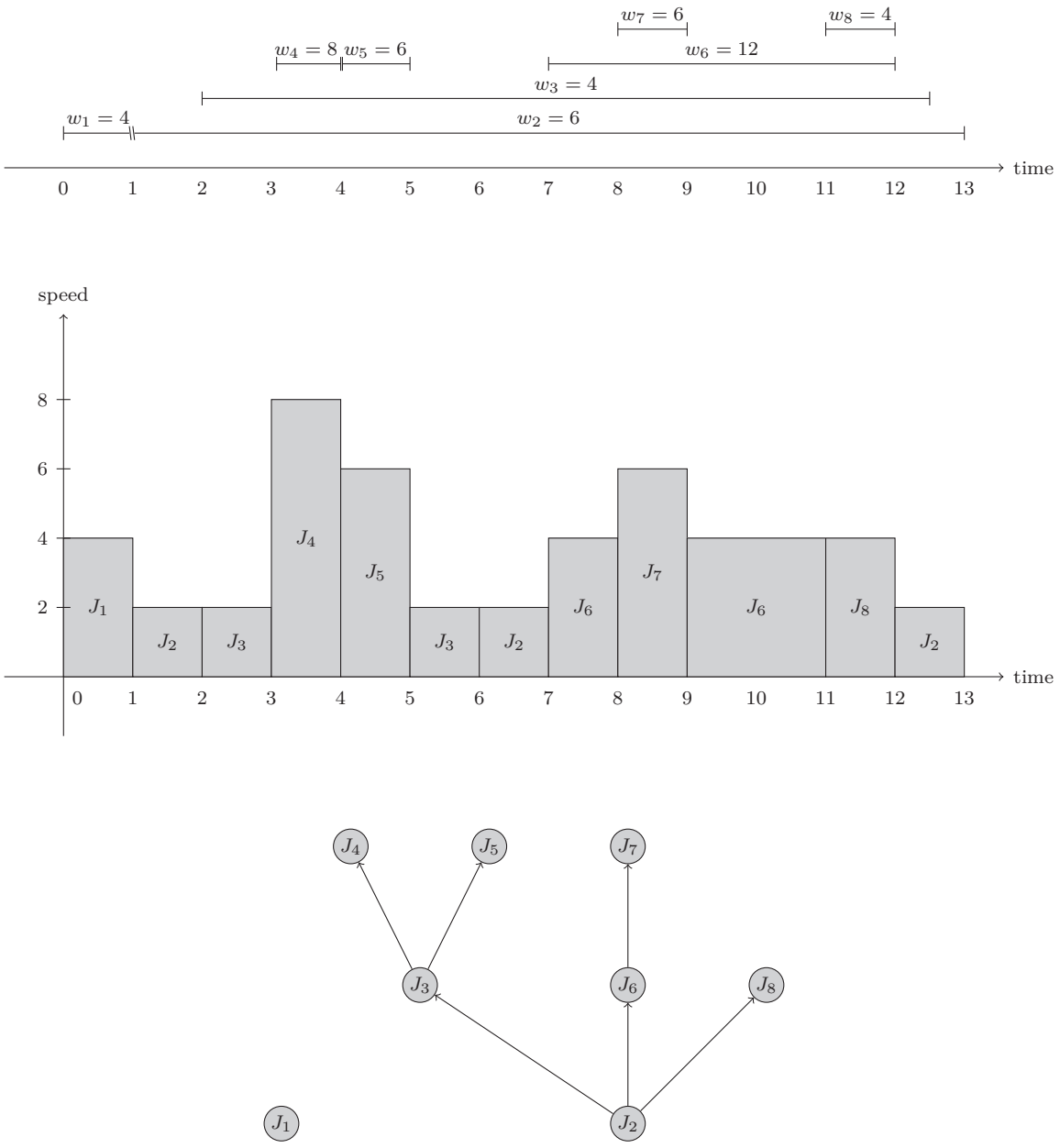


Figure 2.2: An instance of the problem $\mathbf{S}, \mathbf{1}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{pmtn}|\mathbf{E}$, the optimal preemptive schedule produced by the Algorithm 2.1 and its tree representation.

amount of work w_j, a release date r_j and a deadline d_j. In a feasible schedule, every job $J_j \in \mathcal{J}$ is executed entirely during the interval $[r_j, d_j)$.

2.2.1 From Single-Processor Preemptive Schedules

In the following theorem, we show that, for general instances, the ratio between the energy consumption of an optimal non-preemptive schedule to the energy consumption of an optimal preemptive one can be very large.

Theorem 2.1. *The ratio of the energy consumption of an optimal non-preemptive schedule to the energy consumption of an optimal preemptive schedule of the single-processor energy minimization problem can be* $\Omega(n^{\alpha-1})$.

Proof. Consider the instance consisting of $n-1$ unit-work jobs $J_1, J_2, \ldots, J_{n-1}$ and the job J_n of work equal to n. Each job J_j, $1 \leq j \leq n-1$, has release date $r_j = 2j-1$ and deadline $d_j = 2j$, while $r_n = 0$ and $d_n = 2n-1$.

The optimal preemptive schedule $\mathcal{S}^*_{pr}$ (see Figure 2.3) for this instance assigns to all jobs a speed equal to one. Each job J_j, $1 \leq j \leq n-1$, is executed during its whole active interval, while J_n is executed during the remaining n unit length intervals. The total energy consumption of this schedule is

$$E^*_{pr} = (n-1) \cdot 1^\alpha + n \cdot 1^\alpha$$

An optimal non-preemptive schedule $\mathcal{S}^*_{npr}$ for this instance (see Figure 2.3) assigns a speed $\frac{n+2}{3}$ to jobs J_1, J_n and J_2 and schedules them non-preemptively in this order during the interval $[1, 4]$. Moreover, in $\mathcal{S}^*_{npr}$, each job J_j, $3 \leq j \leq n-1$, is assigned a speed equal to one and it is executed during its whole active interval. The total energy consumption of this schedule is

$$E^*_{npr} = 3 \cdot \left(\frac{n+2}{3}\right)^\alpha + (n-3) \cdot 1^\alpha$$

Therefore, we have that

$$\frac{E^*_{npr}}{E^*_{pr}} = \frac{3 \cdot (\frac{n+2}{3})^\alpha + (n-3) \cdot 1^\alpha}{(n-1) \cdot 1^\alpha + n \cdot 1^\alpha} = \Omega(n^{\alpha-1})$$

□

Now, we present an approximation algorithm, whose ratio depends on $w_{\max}$ and $w_{\min}$. In the case where all jobs have equal works, this algorithm achieves a 2^α-approximation ratio. The main idea of our algorithm is to transform the optimal preemptive schedule $\mathcal{S}^*_{pr}$ created by the Algorithm 2.1 into a non-preemptive schedule $\mathcal{S}_{npr}$, based on the corresponding graph $T = (V, A)$ of $\mathcal{S}^*_{pr}$ as it is defined in Section 2.1. More specifically, the jobs are scheduled in three phases depending on the number (zero, one or at least two) of their children in T. A formal description of our algorithm is given in Algorithm 2.2.

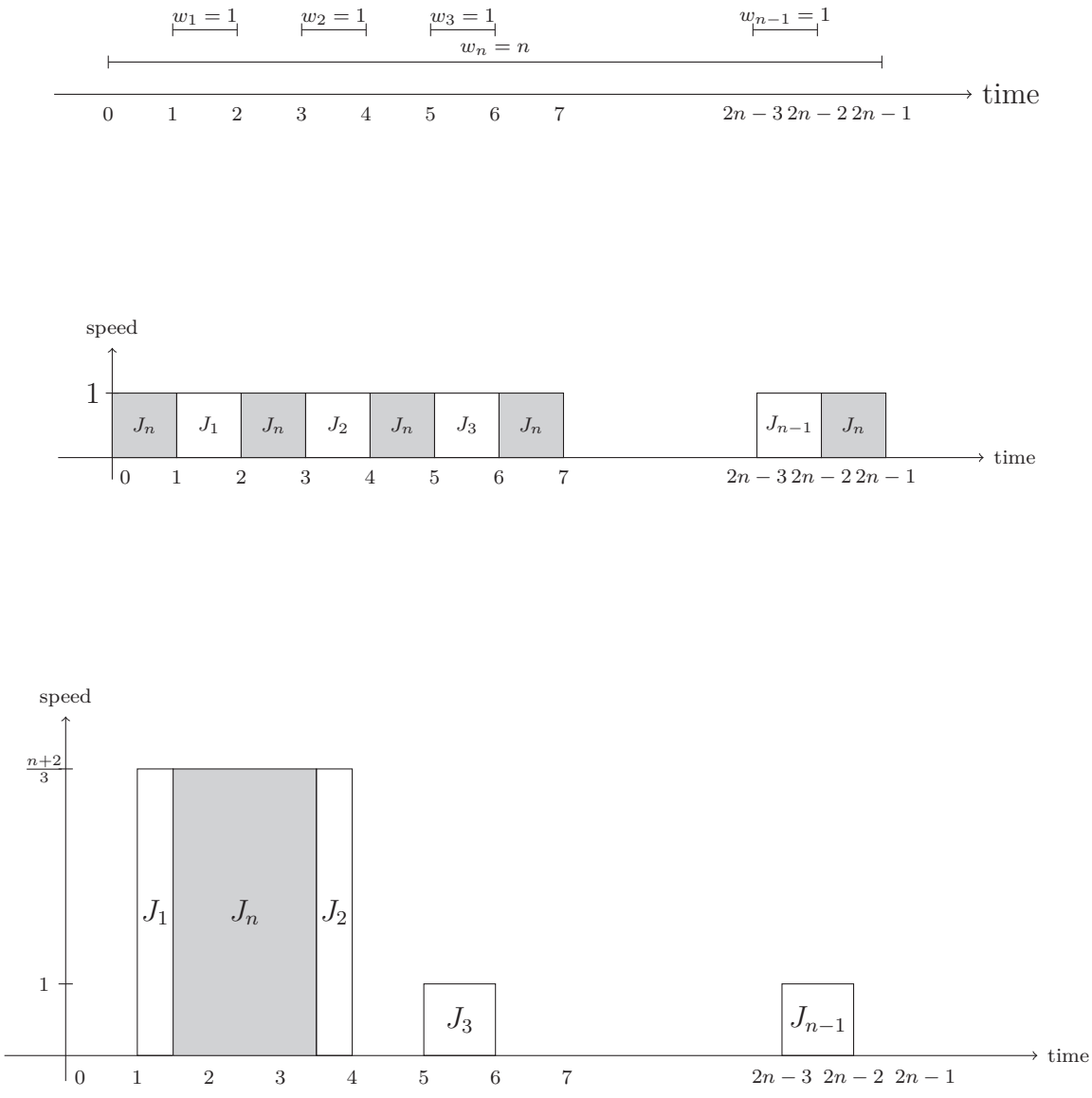


Figure 2.3: An instance for which the ratio between the energy consumption of an optimal non-preemptive schedule and an optimal preemptive schedule is $\Omega(n^{\alpha-1})$. The first schedule is the optimal preemptive one while the second is the optimal non-preemptive one.

Algorithm 2.2

1: Apply the Algorithm 2.1 to create an optimal preemptive schedule $\mathcal{S}^*_{pr}$ and construct the tree representation $T = (V, A)$ of $\mathcal{S}^*_{pr}$.
2: **for** each job J_j with $n_j = 1$ **do**
3: Schedule non-preemptively the whole work of J_j in the biggest interval where a part of J_j is executed in S^*_{pr}.
4: Mark all the remaining jobs as unlabeled.
5: **for** each remaining non-leaf job J_j **do**
6: Find an unlabeled leaf job $J_{j'}$ in T_j and label $J_{j'}$.
7: Schedule non-preemptively J_j and $J_{j'}$ with the same speed in the interval where $J_{j'}$ is executed in $\mathcal{S}^*_{pr}$.
8: Schedule the remaining leaf jobs as in $\mathcal{S}^*_{pr}$.

Theorem 2.2. *Algorithm 2.2 achieves an approximation ratio of* $(1 + \frac{w_{\max}}{w_{\min}})^\alpha$ *for the problem* $\mathbf{S}, \mathbf{1}|\mathbf{r_j}, \mathbf{d_j}|\mathbf{E}$.

Proof. Consider first the jobs with exactly one child in T. By Lemma 2.2, every such job J_j is preempted at most once in $\mathcal{S}^*_{pr}$ and, hence, it is executed into at most two disjoint maximal intervals in $\mathcal{S}^*_{pr}$. In $\mathcal{S}_{npr}$, the whole work of J_j is scheduled in the largest of these two intervals. Thus, the speed of J_j in $\mathcal{S}_{npr}$ is at most twice the speed of J_j in $\mathcal{S}^*_{pr}$. Therefore, for any job J_j with $n_j = 1$ it holds that $E_{npr,j} \leq 2^{\alpha-1} \cdot E^*_{pr,j}$, where $E_{npr,j}$ and $E^*_{pr,j}$ is the energy consumption of J_j in $\mathcal{S}_{npr}$ and $\mathcal{S}^*_{pr}$, respectively.

Consider now the remaining non-leaf jobs. As for each such job J_j it holds that $n_j \geq 2$, in the subtree T_j the number of non-leaf jobs with $n_j \geq 2$ is smaller than the number of leaf jobs. Hence, we can create an one-to-one assignment of the non-leaf jobs with $n_j \geq 2$ to leaf jobs such that each non-leaf job J_j is assigned to a different leaf job $J_{j'} \in T_j$.

Consider a non-leaf job J_j with $n_j \geq 2$ and its assigned leaf job $J_{j'} \in T_j$. Recall that leaf jobs are executed non-preemptively in $\mathcal{S}^*_{pr}$. Let I be the interval in which $J_{j'}$ is executed in $\mathcal{S}^*_{pr}$. Hence, the speed of $J_{j'}$ in $\mathcal{S}^*_{pr}$ is $s^*_{pr,j'} = \frac{w_{j'}}{|I|}$ and its energy consumption is $E^*_{pr,j'} = w_{j'}(s^*_{pr,j'})^{\alpha-1}$. In $\mathcal{S}_{npr}$ both J_j and $J_{j'}$ are executed during I with speed $s_{npr,j} = s_{npr,j'} = \frac{w_j + w_{j'}}{|I|}$. Thus, the energy consumed for J_j and $J_{j'}$ in $\mathcal{S}_{npr}$ is

$$\begin{aligned}
E_{npr,j} + E_{npr,j'} &= w_j(s_{npr,j})^{\alpha-1} + w_{j'}(s_{npr,j'})^{\alpha-1} \\
&= (w_j + w_{j'}) \left(\frac{w_j + w_{j'}}{|I|}\right)^{\alpha-1} \\
&= (w_j + w_{j'})^\alpha \left(\frac{s^*_{pr,j'}}{w_{j'}}\right)^{\alpha-1} \\
&= \left(\frac{w_j + w_{j'}}{w_{j'}}\right)^\alpha \cdot w_{j'}(s^*_{pr,j'})^{\alpha-1} \\
&= \left(\frac{w_j + w_{j'}}{w_{j'}}\right)^\alpha \cdot E^*_{pr,j'} \\
&\leq \left(\frac{w_{\max} + w_{\min}}{w_{\min}}\right)^\alpha \cdot (E^*_{pr,j} + E^*_{pr,j'})
\end{aligned}$$

Moreover, note that $I \subseteq [r_j, d_j)$ and hence $\mathcal{S}_{npr}$ is a feasible schedule.

Finally, for each remaining leaf job J_j, it holds that $E_{npr,j} = E^*_{pr,j}$, concluding the proof of the theorem. □

When all jobs have equal work to execute, we get the following corollary.

Corollary 2.1. *Algorithm 2.2 is 2^α-approximate for* $\mathbf{S}, \mathbf{1}|\mathbf{w_j} = \mathbf{w}, \mathbf{r_j}, \mathbf{d_j}|\mathbf{E}$.

2.2.2 From Multiprocessor Non-Migratory Preemptive Schedules

Next, we present a $2^{\alpha-1}\rho$ approximation algorithm for the non-preemptive problem $\mathbf{S}, \mathbf{1}|\mathbf{r_j}, \mathbf{d_j}|\mathbf{E}$ which uses, as a black box, a ρ-approximation algorithm for the multiprocessor preemptive problem $\mathbf{S}, \mathbf{P}|\mathbf{r_{i,j}}, \mathbf{d_{i,j}}, \mathbf{pmtn}|\mathbf{E}$.

Our algorithm applies a first a transformation to the initial instance. Note that this transformation was first introduced in an algorithm of Antoniadis et al. [9] for the same problem. Then, we give a transformation to the heterogeneous multiprocessor speed-scaling problem without migrations.

We consider the time points $t_0, t_1, t_2, \ldots, t_\tau, t_{\tau+1}$ as follows. Let t_1 be the smallest deadline of any job in $\mathcal{J}$, i.e. $t_1 = \min_{J_j \in \mathcal{J}}\{d_j\}$. Let $\mathcal{R}_1 \subseteq \mathcal{J}$ be the subset of jobs which are released before t_1, i.e. $\mathcal{R}_1 = \{J_j \in \mathcal{J} : r_j < t_1\}$. Next, we set $t_2 = \min_{J_j \in \mathcal{J} \setminus \mathcal{R}_1}\{d_j\}$ and $\mathcal{R}_2 = \{J_j \in \mathcal{J} : t_1 \leq r_j < t_2\}$, and we continue this procedure until all jobs are assigned into a subset of jobs. Let τ be the number of subsets of jobs that have been created. Moreover, let $t_0 = \min_{J_j \in \mathcal{J}}\{r_j\}$ and $t_{\tau+1} = \max_{J_j \in \mathcal{J}}\{d_j\}$. The way we define the time points is depicted in the Figure 2.4.

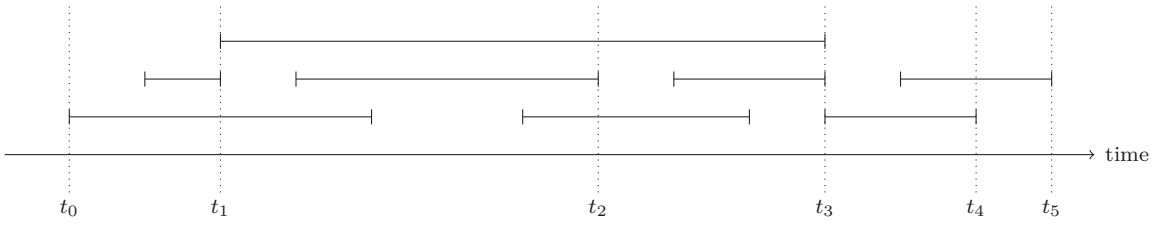


Figure 2.4: An instance of the non-preemptive problem and the time points of the initial transformation.

Consider the intervals $I_\ell = [t_{\ell-1}, t_\ell)$, $1 \leq \ell \leq \tau + 1$. We say that the job $J_j \in \mathcal{J}$ is partially active during the interval I_ℓ if $I_\ell \cup [r_j, d_j) \neq \emptyset$. Let $\mathcal{I}$ be the set of all the intervals I_ℓ. We denote by $\mathcal{H}_j$ the set of intervals I_i in which the job $J_j \in \mathcal{J}$ is partially active, i.e. $\mathcal{H}_j = \{I_\ell \in \mathcal{I} : I_\ell \cup [r_j, d_j) \neq \emptyset\}$. For some intervals in $\mathcal{H}_j$, J_j is active during the whole interval, while in at most two of them it is active during a part of the interval. We consider now the non-preemptive problem in which the execution of J_j should take place into exactly one interval $I_\ell \in \mathcal{H}_j$. Note that the execution of J_j should respect its release date and its deadline.

Lemma 2.3. *Let $\mathcal{S}$ be an optimal non-preemptive schedule for the problem in which the execution of each job $J_j \in \mathcal{J}$ should take place into exactly one interval $I_\ell \in \mathcal{H}_j$. Moreover, let $\mathcal{S}^*$ be the optimal schedule for our original problem. We denote by E and OPT the energy consumption of $\mathcal{S}$ and $\mathcal{S}^*$, respectively. Then, it holds that $E \leq 2^{\alpha-1}OPT$.*

Proof. In order to get a relation between the energy consumption of the schedules $\mathcal{S}$ and $\mathcal{S}^*$, consider first a job $J_j \in \mathcal{R}_\ell$ which can be feasibly executed in more than one intervals, i.e. $|\mathcal{H}_j| \geq 2$. By definition, it holds that $t_{\ell-1} \leq r_j < t_\ell$. Moreover, let $t_{\ell'-1} \leq d_j < t_{\ell'}$, for some $I_{\ell'}$ such that $\ell < \ell'$. Furthermore, consider an interval I_k, $\ell \leq k < \ell'$, and let $J_{j'} \in \mathcal{R}_k$ be the job whose deadline defines t_k, i.e. $d_{j'} = t_k$. By the definition of $\mathcal{R}_k$ and the way we define the time points t_{k-1} and t_k, it must be the case that $t_{k-1} \leq r_{j'} < t_k$. Hence, although J_j might be active at both times t_{k-1} and t_k, its execution in $\mathcal{S}^*$ cannot include both of them; otherwise $J_{j'}$ could not be feasibly executed as $t_{k-1} \leq r_{j'} < d_{j'} = t_k$. Thus, in $\mathcal{S}^*$ the execution of any job cannot include more than two times t_ℓ and $t_{\ell+1}$. Therefore, in $\mathcal{S}^*$, a job cannot be scheduled into more than two consecutive intervals $[t_{\ell-1}, t_\ell)$ and $[t_\ell, t_{\ell+1})$.

Starting from $\mathcal{S}^*$, we create a feasible non-preemptive schedule $\mathcal{S}'$ for the problem in which the execution of each job $J_j \in \mathcal{J}$ takes place into exactly one interval $I_\ell \in \mathcal{H}_j$ by respecting its release date and its deadline. In order to do this, consider a job $J_j \in \mathcal{J}$ which is executed into two intervals I_ℓ and $I_{\ell+1}$ in $\mathcal{S}^*$. Let $p_{j,\ell}$ and $p_{j,\ell+1}$ be the execution time of J_j into I_ℓ and $I_{\ell+1}$, respectively. Assume, without loss of generality, that $p_{j,\ell} \geq p_{j,\ell+1}$. In $\mathcal{S}$, we execute the whole work of J_j during I_ℓ such that its execution takes exactly $\frac{(p_{j,\ell}+p_{j,\ell+1})}{2}$ time. In order to do this, we just have to increase the speed s_j that J_j had in $\mathcal{S}^*$ by at most a factor of 2. Hence, the energy consumption of J_j in $\mathcal{S}^*$ was $(p_{j,\ell} + p_{j,\ell+1})s_j^\alpha$, while in $\mathcal{S}'$ is $\frac{(p_{j,\ell}+p_{j,\ell+1})}{2}(2s_j)^\alpha$. By summing up for all jobs we get that the energy consumption E' of the schedule $\mathcal{S}'$ satisfies $E' \leq 2^{\alpha-1}OPT$. Thus, $E \leq 2^{\alpha-1}OPT$. □

Next, we describe how to pass from the transformed problem to the heterogeneous multiprocessor speed-scaling problem without migrations $\mathbf{S}, \mathbf{P}|\mathbf{r_{i,j}}, \mathbf{d_{i,j}}, \mathbf{pmtn}|\mathbf{E}$. For each interval I_i, $1 \leq i \leq \tau+1$, we create a processor P_i. For each job $J_j \in \mathcal{J}$ which is partially active in the interval I_i, $1 \leq i \leq \tau+1$, we set (i) $r_{i,j} = 0$ if $r_j \leq t_{i-1}$ or $r_{i,j} = r_j - t_{i-1}$ if $r_j > t_{i-1}$, (ii) $d_{i,j} = t_i - t_{i-1}$ if $d_j > t_i$ or $r_{i,j} = d_j - t_{i-1}$ if $d_j \leq t_i$. Note that we keep the same amount of work w_j for each job $J_j \in \mathcal{J}$.

Next, we apply an approximation algorithm for $\mathbf{S}, \mathbf{P}|\mathbf{r_{i,j}}, \mathbf{d_{i,j}}, \mathbf{pmtn}|\mathbf{E}$. This algorithm will create a preemptive schedule $\mathcal{S}$. However, we can transform $\mathcal{S}$ into a non-preemptive schedule $\mathcal{S}'$ of the same energy consumption. To see this, observe that in each processor P_i, $1 \leq i \leq \tau+1$, each job $J_j \in \mathcal{J}$ has $r_{i,j} = 0$ or $d_{i,j} = t_i - t_{i-1}$. Hence, by applying the Earliest Deadline First policy to each processor separately we can get the feasible non-preemptive schedule $\mathcal{S}'$.

Theorem 2.3. *Given a ρ-approximation algorithm for the multiprocessor problem $\mathbf{S}, \mathbf{P}|\mathbf{r_{i,j}}, \mathbf{d_{i,j}}, \mathbf{pmtn}|\mathbf{E}$ the single-processor speed-scaling problem without preemptions can be approximated within a factor of $2^{\alpha-1}\rho$.*

2.3 Maximum Lateness Minimization

In this section, we consider non-preemptive problems of minimizing the energy and the maximum lateness of a set of jobs. Initially, we consider the offline problem $\mathbf{S, 1|r_j|L_{max}(E)}$ of minimizing the maximum lateness under a budget of energy and we propose a polynomial time algorithm for the special case where the jobs are released at the same time. Moreover, we show that the problem is $\mathcal{NP}$-hard when the release dates of the jobs are arbitrary. Next, we move our attention to the online problem $\mathbf{S, 1|r_j|L_{max} + \beta E}$, where the objective is to minimize a linear combination of the maximum lateness and the energy, and we present a 2-competitive algorithm.

In the maximum lateness minimization problems, there is a set of n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ which have to be scheduled by a single processor. A job $J_j \in \mathcal{J}$ comes with an amount of work w_j, a release date r_j and a delivery time q_j. The release date r_j corresponds to the arrival time of J_j. In a given a schedule $\mathcal{S}$, let C_j be the completion time of J_j. Then the lateness of J_j is defined as $L_j = C_j + q_j$ in $\mathcal{S}$. In the budget problem, our objective is to find a schedule such that the maximum lateness among the jobs, i.e. $L_{max} = \max_{J_j \in \mathcal{J}}\{L_j\}$, is minimized and the total energy consumption of the schedule does not exceed an energy budget equal to E. In the aggregate problem, we want to minimize a linear combination of the maximum lateness and the energy, i.e. $L_{max} + \beta E$. In the offline setting, all the information of the problem's instance are known in advance. On the other hand, in the online setting, the existence of a job J_j and its parameters are known only when the job has arrived, that is at its release date.

2.3.1 Offline

We begin our study for the problem of minimizing the maximum lateness in the offline setting.

Common Release Date

In the following, we describe an optimal algorithm for the problem $\mathbf{S, 1||L_{max}(E)}$. In the beginning, we present a convex programming formulation for the problem. This formulation implies directly that the problem is polynomially solvable as convex programs can be solved in polynomial time by applying the Ellipsoid algorithm. Then, we apply the well-known KKT conditions to the convex program and we deduce some necessary and sufficient properties that any feasible solution of the convex program must satisfy in order to be optimal. Based on these properties, we derive a faster combinatorial algorithm.

A convex programming formulation of the problem stems from two basic properties of an optimal schedule. First, because of the convexity of the speed to power function, each job $J_j \in \mathcal{J}$ runs with constant speed s_j. Second, in any optimal schedule, the jobs are scheduled according to the EDD (Earliest Due Date First) rule, or, equivalently, in non-increasing order of their delivery times; this can be easily shown by a standard exchange argument. Hence, we propose the following formulation where all jobs are considered to be released at time zero and numbered according to the EDD order.

$$\min L$$

$$C_j + q_j \leq L \qquad 2 \leq j \leq n \tag{2.1}$$

$$\frac{w_1}{s_1} \leq C_1 \tag{2.2}$$

$$C_{j-1} + \frac{w_j}{s_j} \leq C_j \qquad 2 \leq j \leq n \tag{2.3}$$

$$\sum_{j=1}^{n} w_j s_j^{\alpha-1} \leq E \tag{2.4}$$

$$L, C_j, s_j \geq 0 \qquad 1 \leq j \leq n \tag{2.5}$$

Our objective is to minimize the maximum lateness L. The constraints (2.1) ensure that the lateness of each job is at most L, the constraints (2.2) and (2.3) enforce the jobs to be scheduled according to the EDD rule in non-overlapping time intervals, the constraint (2.4) does not allow to exceed the given energy budget E and the constraints (2.5) ensure that the maximum lateness, the completion times and the speeds of jobs are non-negative. The constraint (2.4) is convex for $\alpha > 2$ while all other constraints and the objective function are linear. Thus, our mathematical program is indeed convex.

This convex program already implies a polynomial algorithm for our problem, as convex programs can be solved to arbitrary precision by the Ellipsoid algorithm [55]. However, we will exploit this convex program to derive a faster combinatorial algorithm.

In what follows we deduce a number of structural properties of an optimal schedule by applying the KKT conditions to the above convex program. The general form of the KKT conditions can be found in the Appendix A. Note that the jobs are indexed $J_1, J_2, \ldots, J_n$ according to the EDD order. That is, for any couple of jobs $J_j, J_{j'} \in \mathcal{J}$ such that $j < j'$, it must be the case that $q_j \geq q_{j'}$. Furthermore, in a given schedule $\mathcal{S}$, we say that the job J_j is critical if it attains the maximum lateness of the schedule, i.e. $L_j = L_{max}$.

Lemma 2.4. *For the maximum lateness problem with an energy budget E, there is always an optimal schedule that satisfies all the following properties.*
(i) Each job J_j runs at a constant speed s_j.
(ii) Jobs are scheduled according to the EDD rule.
(iii) There are no idle periods in the schedule.
(iv) The last job is critical, i.e. $L_n = L_{max}$.
(v) Every non-critical job J_j has equal speed with the job J_{j+1}, i.e. $s_j = s_{j+1}$.
(vi) Jobs are executed in non-increasing speeds, i.e. $s_j \geq s_{j+1}$.
(vii) All the energy budget is consumed.

Proof. We associate to each set of constraints from (2.1) up to (2.4) the dual variables $\lambda_i, \mu_1, \mu_i, \xi$, respectively. Without loss of generality, the variables L, C_i and s_i are positive and, by the complementary slackness conditions, the dual variables associated to the constraints (2.5) are equal to zero in any optimal solution of the convex program.

Stationarity conditions give that

$$\begin{aligned} \nabla L + \sum_{j=1}^{n} \lambda_j \nabla(C_j + q_j - L) + \mu_1 \nabla(\frac{w_1}{s_1} - C_1) \\ + \sum_{j=2}^{n} \mu_j \nabla(C_{j-1} + \frac{w_j}{s_j} - C_j) + \xi \nabla(\sum_{j=1}^{n} w_j s_j^{a-1} - E) &= 0 \Rightarrow \\ (1 - \sum_{j=1}^{n} \lambda_j)\nabla L + \sum_{j=1}^{n-1} (\lambda_j - \mu_j + \mu_{j+1})\nabla C_j \\ + (\lambda_n - \mu_n)\nabla C_n + \sum_{j=1}^{n} (-\mu_j w_j s_j^{-2} + (a-1)\xi w_j s_j^{a-2})\nabla s_j &= 0 \end{aligned}$$

Therefore, we get equivalently that

$$\sum_{j=1}^{n} \lambda_j = 1 \tag{2.6}$$

$$\lambda_j = \mu_j - \mu_{j+1} \qquad 1 \le j \le n-1 \tag{2.7}$$

$$\lambda_n = \mu_n \tag{2.8}$$

$$(\alpha - 1)\xi = \frac{\mu_j}{s_j^{\alpha}} \qquad 1 \le j \le n \tag{2.9}$$

Moreover, complementary slackness conditions give that

$$\lambda_j(C_j + q_j - L) = 0 \qquad 1 \le j \le n \tag{2.10}$$

$$\mu_1(\frac{w_1}{s_1} - C_1) = 0 \tag{2.11}$$

$$\mu_j(C_{j-1} + \frac{w_j}{s_j} - C_j) = 0 \qquad 2 \le j \le n \tag{2.12}$$

$$\xi\left(\sum_{j=1}^{n} w_j s_j^{\alpha-1} - E\right) = 0 \tag{2.13}$$

The fact that (i) and (ii) are satisfied by an optimal schedule has been already discussed above. We claim that $\xi \neq 0$. Assume for contradiction that $\xi = 0$. Then, by (2.9), we get that $\mu_j = 0$ for each $1 \le j \le n$. This, combined with (2.7) and (2.8) yields that $\sum_{j=1}^{n} \lambda_j = 0$, which is a contradiction because of (2.6). Since $\xi \neq 0$, we get by (2.9) that $\mu_j \neq 0$ for each $1 \le j \le n$. Then, equations (2.11) and (2.12) give that there is no idle time in an optimal schedule since it must be the case that $C_1 = \frac{w_1}{s_1}$ and $C_j = C_{j-1} + \frac{w_j}{s_j}$, for $2 \le j \le n$. Since $\xi \neq 0$, by (2.9), it follows that $\mu_n \neq 0$ and finally, because of (2.8), $\lambda_n \neq 0$. So, the last job to finish is always a critical job, by (2.10).

Note that for every non-critical job J_j, it holds that $C_j + q_j < L$ and (2.10) implies that $\lambda_j = 0$ for every such job. Hence, if a job J_j is non-critical, then $\lambda_j = 0 \Rightarrow \mu_j = \mu_{j+1} \Rightarrow s_j = s_{j+1}$, by (2.7) and (2.9), respectively. By the dual feasibility conditions and the equations (2.7) and (2.9) we get, respectively, that $\lambda_j \ge 0 \Rightarrow \mu_j \ge \mu_{j+1} \Rightarrow s_j \ge s_{j+1}$. Thus, the jobs are executed with non-increasing speeds. If the energy budget is not entirely consumed, then by (2.13), $\xi = 0$, which is a contradiction, since, as we have already proved, $\xi \neq 0$.

Note that, given any feasible schedule that satisfies the properties of the lemma, that is a feasible solution to the convex program, we can give values to the dual variables such that the KKT conditions are satisfied. Therefore, any schedule satisfying the properties is optimal. □

We refer to any schedule satisfying the properties of Lemma 2.4 as a *regular* schedule. By (i, j) we denote a sequence of consecutive jobs $J_i, J_{i+1}, \ldots, J_j$. Any regular schedule can be partitioned into groups of jobs, of the form (i, j), where the jobs J_{i-1} and J_j are critical and the jobs $J_i, J_{i+1}, \ldots, J_{j-1}$ are not. By Lemma 2.4, all jobs of such a group are executed at the same speed. We denote this common speed by s_j and the total amount of work of jobs in (i, j) by $w(i, j) = \sum_{k=i}^{j} w_k$. Then, the next corollary follows easily from Lemma 2.4.

Corollary 2.2. *Let J_i, J_j, be two consecutive critical jobs of a regular schedule. The speed of each job in the group $(i + 1, j)$ is equal to $s_j = \frac{w(i+1,j)}{q_i - q_j}$.*

Proof. Since the jobs J_i and J_j are critical, we have that $L_i = L_j = L_{max}$. Thus, $C_j - C_i = q_i - q_j$. Because of the Lemma 2.4, the jobs in $(i + 1, j)$ are executed with constant speed s_j between C_i and C_j. As there are no idle periods in the schedule, the processing time of the jobs in $(i + 1, j)$ is equal to $C_j - C_i$. Hence,

$$\frac{w(i+1,j)}{s_j} = C_j - C_i \Rightarrow s_j = \frac{w(i+1,j)}{q_i - q_j}$$

□

So far, we have derived a clear image of the structure of any regular optimal schedule for $\mathbf{S, 1||L_{max}(E)}$. Next, we propose Algorithm 2.3 which constructs such a schedule in polynomial time. Note that a regular schedule is fully specified by the speeds of the jobs. The rough idea of our algorithm is the following: First, it constructs a preliminary schedule by finding groups of jobs running in non-increasing speeds without taking care of the energy consumption. Second, the algorithm manages the energy consumption with respect to the energy budget E and determines the final speeds of all jobs. Let E' be the energy consumption of the current schedule at any point of the execution of the algorithm.

Algorithm 2.3 starts from job J_n which is always a critical job and considers all jobs but the first, in reverse order. When a job J_i, $2 \leq i \leq n$, is considered for the first time, its speed s_i is set according to Corollary 2.2, assuming that jobs J_{i-1} and J_i are critical. If $s_i \geq s_j$, for $i + 1 \leq j \leq n$, then s_i is called *eligible* speed and it is assigned to job J_i. If this speed is not eligible, J_i is a non-critical job and it is merged with the J_{i+1}'s group. More specifically, if J_c is the last job of this group, then the speeds of jobs $J_i, J_{i+1}, \ldots, J_c$ are calculated by applying Corollary 2.2, assuming that J_{i-1} and J_c are critical while $J_i, J_{i+1}, \ldots, J_{c-1}$ are not. Next, the algorithm examines whether the new value of s_i is eligible. If this is the case, then it considers the job J_{i-1}. Otherwise, a further merging, of the J_i's group with the J_{c+1}'s group, is performed, as before. That is, if $J_{c'}$ is the last job of the J_{c+1}'s group, all jobs $J_i, J_{i+1}, \ldots, J_{c'}$ are assigned the same speed assuming that jobs J_{i-1} and $J_{c'}$ are critical, while $J_i, J_{i+1}, \ldots, J_{c'-1}$ are not. This

speed, according to the Corollary 2.2, is equal to $s(i,c') = \frac{w(i,c')}{q_{i-1}-q_{c'}}$. Note that the job J_c is no longer critical in this case. This merging procedure is repeated until job J_i is assigned an eligible speed. In a degenerate case, jobs $J_i, J_{i+1}, \ldots, J_n$ are merged into one group. When the algorithm has assigned an eligible speed to all jobs $J_2, J_3, \ldots, J_n$, it sets $s_1 = s_2$ and its first part completes.

Next, Algorithm 2.3 takes into account the available budget of energy E. If $E - E' \geq 0$, the current schedule's energy consumption does not exceed the budget of energy, and the surplus $E - E'$ is assigned to the first job. Otherwise, the current schedule is regular, except that it consumes an amount of energy greater than E. Then, the algorithm reduces the consumed energy until it becomes equal to E. In fact, it decreases the speed of the first group, by merging groups with the first one if necessary. This merging procedure is different from the one of the first part of the algorithm and it is as follows: let J_i be the critical job of maximal index with $s_i = s_1$ in the current schedule. Observe that $s_i > s_{i+1}$. The algorithm sets the speed of jobs $J_1, J_2, \ldots, J_i$ equal to s_{i+1}. This causes a reduction to E' and there are two cases to distinguish: either $E' \leq E$ or $E' > E$. In the first case, the algorithm adds an amount of energy $E - E'$ to jobs $J_1, J_2, \ldots, J_i$ by increasing their speeds uniformly, i.e. so that they are all executed with the same speed. In the second case, at least one further merging step has to be performed. When the algorithm terminates, it is obvious that $E' = E$.

Algorithm 2.3

```
1: Sort the jobs according to the EDD order.
2: for j = n to 2 do
3:    Set s_j assuming that J_j and J_{j-1} are critical.
4:    while s_j is not eligible do
5:       Merge the J_j's group with the next group.
6: Set s_1 = s_2
7: Let E' be the current energy consumption.
8: if E > E' then
9:    Assign energy E - E' to job 1.
10: else
11:   while E < E' do
12:      Set the speed of the first group equal to the speed of the following group.
13:      Update E'.
14:      if E < E' then
15:         Merge the first group with the next one.
16:   Assign E - E' energy uniformly to the first group.
```

Theorem 2.4. *Algorithm 2.3 is optimal for* $\mathbf{S}, \mathbf{1}||\mathbf{L_{max}}(\mathbf{E})$.

Proof. We shall prove that the algorithm satisfies the properties of Lemma 2.4, i.e. it produces a regular schedule. For convenience, we distinguish two parts in the algorithm: *Part I*, corresponding to lines 1-6 and *Part II*, corresponding to lines 7-16, respectively.

Properties (i)-(ii): The algorithm gives a single constant speed to each job and keeps their initial EDD order.

Property (iii): In Part I, the speeds of jobs are assigned according to Corollary 2.2. Specifically, the algorithm fixes two consecutive critical jobs J_i and J_j, $i < j$, with, potentially, some non-critical jobs between them. Then the speed of the non-critical jobs and the one of the critical job J_j is defined such that there is no idle between the jobs. In Part II, no idle period is added between any jobs.

Properties (iv) - (v): When the speed of job J_n is initialized, this is done by assuming that it is critical. Next, consider the current schedule just after the completion of Part I. This schedule can be partitioned into sequences of jobs, $J_{a+1}, J_{a+2}, \ldots, J_b$, with $a \geq 1$, such that the jobs of each sequence are executed with the same speed which has been assigned by applying Corollary 2.2, assuming that the jobs J_a and J_b are critical. In fact, jobs J_a and J_b attain equal lateness. In order for such a sequence to be a group, we should also prove that all but the last jobs are non-critical while the last job is critical.

Let $J_{a+1}, J_{a+2}, \ldots, J_b$ be a sequence of jobs. We claim that $L_i < L_b$, for $a+1 \leq i \leq b-1$. Assume, by contradiction, that there exists a job J_j, where $a+1 \leq j \leq b-1$, such that $L_j \geq L_b$, or equivalently, $q_j - q_b \geq \sum_{i=j+1}^{b} \frac{w_i}{s_b}$. Since the last job of a sequence attains equal lateness with the last job of the sequence that follows, we have that $L_a = L_b$. This yields that $q_a - q_b = \sum_{i=a+1}^{b} \frac{w_i}{s_b}$. Therefore, $q_a - q_j \leq \sum_{i=a+1}^{j} \frac{w_i}{s_b}$.

Obviously, for any job J_i, $a+1 \leq i \leq b-1$, we must have a speed $s_i > \frac{w_i}{q_{i-1}-q_i}$, since otherwise, it wouldn't have been merged with another group. That is, $q_{i-1} - q_i > \frac{w_i}{s_i}$. If we sum the last inequalities for $a+1 \leq i \leq j$, we get that $q_a - q_j > \sum_{i=a+1}^{j} \frac{w_i}{s_b}$, a contradiction.

At this point, we have showed that when Part I completes, if a job J_i, $2 \leq i \leq n$, is critical, then it must be the right extremity of a sequence. Moreover, among all jobs $J_2, J_3, \ldots, J_n$, the last jobs of all sequences, including job J_n, attain equal lateness and the remaining jobs attain smaller lateness. In addition, job J_1 attains equal lateness with the last job of the sequence that follows. Recall that, at this point, we set $s_1 = s_2$. Job J_1 would have equal lateness with the last job of the sequence that follows for any $s_1 > 0$ since the speed of the second group is set by applying Corollary 2.2, assuming that J_1 is critical. So, at the end of Part I, job J_1, job J_n and every last job of a sequence are critical. Therefore, after Part I finishes, *Properties (iv)* and *(v)* hold.

In Part II, if no merging step is performed, then the processing time of job J_1 is decreased by some $x \geq 0$ and its lateness decreases by x, while the processing times and speeds of the other jobs are not modified. So, the lateness of every other job also decreases by x. Hence, the *Properties (iv)* and *(v)* hold.

If at least one merging step is performed, then the speed of the jobs in the first group decreases and their processing time increases. Then, in the first group, every non-critical job J_i has equal speed with the job J_{i+1} that follows, while the speeds of the jobs in other groups remain unchanged. Now, let x_i be the total increase in the processing time of job J_i, $1 \leq i \leq n$. Note that this quantity is positive only for jobs belonging to the first group of the current schedule. Then, the lateness of any job J_i, $1 \leq i \leq n$, increases by $\sum_{j=1}^{i} x_j$; if J_{c_1} is the critical job of the first group, it remains critical after the merging step since its lateness and the lateness of every other job that follows, increases by the same quantity, equal to $\sum_{j=1}^{c_1} x_j$. Note, that if a further merging step is performed, we consider the first two groups as one group. Moreover, the lateness of any job increases by no more than the increase of the lateness of job J_n, and thus, in the final schedule, job

J_n remains critical and *Property (iv)* holds. Furthermore, each non-critical job has equal speed with the job that follows and *Property (v)* holds as well.

Property (vi): At the end of Part I, the speeds of jobs are non-increasing since otherwise, a merging step would be performed. Moreover, during Part II, no speed of a job becomes less than the speed of a subsequent job.

Property (vii): Recall that E' is the total energy consumed when Part I completes. If E' is less than the energy budget, then the energy of the first job increases until the schedule consumes exactly E units of energy, while if E' is greater than the energy budget E, then the energy consumption of the schedule is gradually decreased until it becomes equal to E.

Let us now consider the complexity of the algorithm. Initially, jobs are sorted according to the EDD rule in $O(n \log n)$ time. The first part of the algorithm may take $O(n^2)$ time since each merging step takes $O(n)$ time and there can be $O(n)$ merging steps. Also, the algorithm's second part takes $O(n^2)$ time since the speed of each job may change at most $O(n)$ times. Therefore, the overall complexity of the algorithm is $O(n^2)$. □

Arbitrary Release Dates

We now consider the budget variant of the maximum lateness problem, where the jobs have arbitrary release dates, i.e. $\mathbf{S, 1|r_j|L_{max}(E)}$, and we show that it is strongly $\mathcal{NP}$-hard. In order to establish this $\mathcal{NP}$-hardness result, we present a reduction from 3-PARTITION which is known to be strongly $\mathcal{NP}$-hard [35].

In 3-PARTITION, we are given a positive integer B and a set of $3n$ positive integers $A = \{a_1, a_2, \dots a_{3n}\}$, where $B/4 < a_j < B/2$ and $\sum_{a_j \in A} a_j = nB$, and we ask if there exists a partition of A into n disjoint sets $A_1, A_2 \dots, A_n$ such that $\sum_{a_j \in A_k} a_j = B$, for each $1 \leq k \leq n$.

Our reduction is inspired by the $\mathcal{NP}$-hardness proof for the classical problem $\mathbf{1|r_j|L_{max}}$ [35], where we are given a set of jobs $\mathcal{J}$ with each job $J_j \in \mathcal{J}$ having a release date r_j, a delivery time q_j and a processing time p_j and we seek a schedule minimizing the maximum lateness. This problem can be viewed as a variant of our problem where the speed of each job is part of the instance. Specifically, we consider that each job J_j has an amount of work $w_j = p_j$ and it is executed at a constant speed $s_j = 1$. Based on this idea, we adapt the existing $\mathcal{NP}$-hardness reduction of $\mathbf{1|r_j|L_{max}}$ by fixing an energy budget, so that all jobs have to be executed with the same speed $s_j = 1$ in order to get a feasible schedule.

Theorem 2.5. *The problem* $\mathbf{S, 1|r_j|L_{max}(E)}$ *is strongly* $\mathcal{NP}$*-hard.*

Proof. We construct an instance of $\mathbf{S, 1|r_j|L_{max}(E)}$ from an instance of 3-PARTITION as follows.

- For each a_j, $1 \leq j \leq 3n$, we create a job J_j with $w_j = a_j$, $r_j = 0$ and $q_j = 0$.
- We introduce $n-1$ gadget jobs, where the gadget job J_j, $3n+1 \leq j \leq 4n-1$, has $w_j = B, r_j = (2j - 6n - 1)B$ and $q_j = (8n - 2j - 1)B$.
- We set $E = (2n-1)B$.

j	w_j	r_j	q_j
1	a_1	0	0
2	a_2	0	0
...	...	...	...
$3n$	a_{3n}	0	0
$3n+1$	B	B	$(2n-3)B$
$3n+2$	B	$3B$	$(2n-5)B$
$3n+3$	B	$5B$	$(2n-7)B$
...	...	...	...
$4n-2$	B	$(2n-5)B$	$3B$
$4n-1$	B	$(2n-3)B$	B

Table 2.1: An instance of $\mathbf{S,1|r_j|L_{max}(E)}$ constructed from an instance of 3-PARTITION.

Our construction is depicted in the Table 2.1.

We claim that there is a feasible schedule $\mathcal{S}$ with $L_{max} = (2n-1)B$ and total energy consumption $E = (2n-1)B$ if and only if there exists a 3-PARTITION of A. For convenience, we denote by $\mathcal{J}$, $\mathcal{G}$ the set of all the jobs and the set of the gadget jobs, respectively.

($\Leftarrow$) For the first direction, assume that $A_1, A_2 \ldots, A_n$ is a partition of A, where $\sum_{a_j \in A_k} a_j = B$, for $1 \leq k \leq n$. Then, consider the schedule $\mathcal{S}$ in which

- each job $J_j \in \mathcal{J} \setminus \mathcal{G}$ corresponding to an integer $a_j \in A_k$, $1 \leq k \leq n$, is scheduled during the time interval $((2k-2)B, (2k-1)B]$,
- each gadget job $J_j \in \mathcal{G}$ is scheduled during $((2j-6n-1)B, (2j-6n)B]$, and
- all jobs are executed at constant speed $s_j = 1$.

Clearly, the schedule $\mathcal{S}$ is feasible and it attains maximum lateness $L_{max} = (2n-1)B$. Its total energy consumption is $E = \sum_{J_j \in \mathcal{J}} w_j s_j^{\alpha-1} = \sum_{J_j \in \mathcal{J}} w_j = (2n-1)B$.

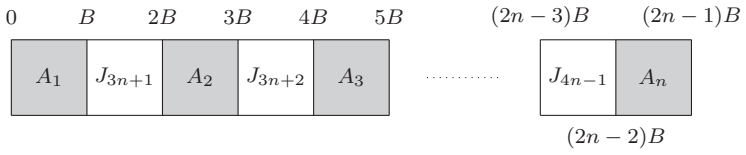


Figure 2.5: The schedule $\mathcal{S}$.

($\Rightarrow$) For the opposite direction, assume that there exists a feasible schedule $\mathcal{S}$ with $L_{max} = (2n-1)B$ and total energy consumption $E = (2n-1)B$. In $\mathcal{S}$, each job J_j, $1 \leq j \leq 3n$, has completion time $C_j \leq (2n-1)B$ and each gadget job J_j, $3n+1 \leq i \leq 4n-1$, has completion time $C_j \leq (2j-6n)B$, since $L_j \leq (2n-1)B$ for every job J_j. For notational convenience, let $W = (2n-1)B$ be the sum of the works of all the jobs.

It must be the case that the makespan of the schedule $\mathcal{S}$ is $C_{max} = (2n-1)B$. To see this, assume for the sake of contradiction that $C_{max} < (2n-1)B$. In this case, due to the convexity of the speed-to-power function, we know that, for the energy consumption $E(\mathcal{S})$ of the schedule $\mathcal{S}$, it would hold that

$$E(\mathcal{S}) \geq C_{max} \left(\frac{W}{C_{max}}\right)^{\alpha} > (2n-1)B$$

which is not possible because the energy is exceeded. With a similar argument, it can be shown that there will be no idle time during $(0, (2n-1)B]$.

Due to the convexity of the speed-to-power function, among the schedules with makespan $C_{max} = (2n-1)B$ which have no idle period during $(0, (2n-1)B]$, only the ones in which all the jobs are executed with speed equal to $s_j = 1$ have energy consumption not greater than $E = (2n-1)$. Clearly, $\mathcal{S}$ must be one of these schedules. Hence, every gadget job $J_j \in \mathcal{G}$ is executed within the whole time interval $((2j-6n-1)B, (2j-6n)B]$ in $\mathcal{S}$.

So far we have shown that every gadget job $J_j \in \mathcal{G}$, spans the time interval $((2j-6n-1)B, (2j-6n)B]$ in $\mathcal{S}$, while the other jobs $J_j \in \mathcal{J} \setminus \mathcal{G}$ span the time intervals $((2k-2)B, (2k-1)B]$, for $1 \leq k \leq n$. Therefore, during any interval $((2k-2)B, (2k-1)B]$, $1 \leq k \leq n$, there will be executed a set of jobs with total amount of work B in $\mathcal{S}$, as every job $J_j \in \mathcal{J}$ is executed with constant speed $s_j = 1$. This execution defines a 3-PARTITION for A. □

2.3.2 Online

Now, we turn our attention to the online version of the maximum lateness objective. Clearly, we do not expect a constant factor competitive algorithm for the budget problem $\mathbf{S, 1|r_j|L_{max}(E)}$. This can be shown by defining an adversarial strategy, such as the one of Bansal et al. [17] for the average completion time objective, which makes any online constant-factor competitive deterministic algorithm run out of energy without completing all the jobs. Therefore, following the approach of Albers et al. [5] for the total flow time objective, we consider the problem of minimizing a linear combination of the maximum lateness and the energy, i.e. $\mathbf{S, 1|r_j|L_{max} + \beta E}$, and we derive a 2-competitive algorithm.

Our algorithm for $\mathbf{S, 1|r_j|L_{max} + \beta E}$ schedules the jobs in a number of phases by repeatedly applying an optimal offline algorithm for the problem $\mathbf{S, 1||L_{max} + \beta E}$. Specifically, the jobs are scheduled in batches and all the jobs of the same batch are scheduled as if they have a common release date. In the following, we first obtain an optimal offline algorithm for $\mathbf{S, 1||L_{max} + \beta E}$ and, then, we present our online algorithm for the problem with release dates.

Optimal Offline Algorithm for $\mathbf{S, 1||L_{max} + \beta E}$

In order to derive an optimal algorithm for the maximum lateness plus weighted energy problem, we follow the same line as for the budget problem. By formulating the problem as a convex program and applying the KKT conditions, we get some necessary and sufficient conditions of optimality for any feasible schedule. Then, we describe an algorithm which always produces a solution satisfying these conditions.

Similarly with the budget problem, there is always an optimal schedule which executes the jobs with respect to the EDD (Earliest Due Date first) rule, i.e. in non-increasing order of delivery times. In what follows, we number the jobs according to the EDD order and, for a given schedule, we say that a job J_j is critical if it attains the maximum lateness L_{max} of the schedule, i.e. $L_j = L_{max}$. Then, the problem $\mathbf{S}, \mathbf{1}||\mathbf{L}_{\max} + \beta\mathbf{E}$ can be formulated as follows.

$$\min L + \beta \sum_{j=1}^{n} w_j s_j^{\alpha-1} \tag{2.14}$$

$$C_j + q_j \leq L \qquad 1 \leq j \leq n \tag{2.15}$$

$$\frac{w_1}{s_1} \leq C_1 \tag{2.16}$$

$$C_{j-1} + \frac{w_j}{s_j} \leq C_j \qquad 2 \leq j \leq n \tag{2.17}$$

$$L, C_j, s_j \geq 0 \qquad 1 \leq j \leq n \tag{2.18}$$

The expression (2.14) is our objective function. Inequality (2.15) ensures that the lateness of each job is no more than the maximum lateness L. The constraints (2.16) and (2.17) enforce that jobs should ordered according to the EDD rule. Finally, the constraints (2.18) ensure the non-negativity of the maximum lateness, the completion times and the speeds of jobs, respectively. Note that the objective function and all the constraints are convex for $\alpha > 2$ and, as a result, the above mathematical program is convex.

By applying the KKT conditions, to the above convex program we get the following lemma whose proof is deferred in the Appendix B because it resembles with the proof of Lemma 2.4.

Lemma 2.5. *There is an optimal schedule for the maximum lateness plus weighted energy problem satisfying the following properties.*
(i) Each job J_j runs at a constant speed s_j.
(ii) Jobs are scheduled according to the EDD rule.
(iii) Jobs are consecutively executed without any idle period.
(iv) The last job is critical, i.e., $L_n = L_{max}$.
(v) Every non-critical job J_j has equal speed with the job J_{j+1}, i.e., $s_j = s_{j+1}$.
(vi) Jobs are executed in non-increasing speeds, i.e., $s_j \geq s_{j+1}$.
(vii) The job executed first runs at speed $s_1 = (\frac{1}{(\alpha-1)\beta})^{\frac{1}{\alpha}}$.

Note that the structure of the optimal schedule for the maximum lateness plus weighted energy problem is almost the same as that of the budget problem with one single difference: the energy consumption is not equal to a fixed value, but it results from the fact that the speed of the first job should always be equal to $(\frac{1}{(\alpha-1)\beta})^{\frac{1}{\alpha}}$. This modification turns both the optimal algorithm and its analysis for the aggregated variant simpler than those of the budget variant.

According to the following lemma, the optimal schedule $\mathcal{S}^*$ for $\mathbf{S}, \mathbf{1}||\mathbf{L}_{\max} + \beta\mathbf{E}$ attains the same maximum lateness with the schedule $\mathcal{S}_c$, in which each job is executed with

constant speed $s_c = (\frac{1}{(\alpha-1)\beta})^{\frac{1}{\alpha}}$. This observation implies that if J_j is the highest-index critical job in $\mathcal{S}_c$, then all jobs $J_1, J_2, \ldots, J_j$ are executed with the speed s_c in $\mathcal{S}^*$.

Lemma 2.6. *Let L_{max} be the maximum lateness of the EDD schedule $\mathcal{S}_c$ that executes each job at a constant speed $s_c = (\frac{1}{(\alpha-1)\beta})^{\frac{1}{\alpha}}$. Moreover, let L^*_{max} be the maximum lateness of an optimal schedule $\mathcal{S}^*$ satisfying the conditions of Lemma 2.5. It must be the case that $L_{max} = L^*_{max}$.*

Proof. We denote by s_j and s^*_j the speed of the job J_j in the schedule $\mathcal{S}_c$ and $\mathcal{S}^*$, respectively. Assume, by contradiction, that $L_{max} \neq L^*_{max}$. First, suppose that $L_{max} > L^*_{max}$. This is possible only if there is at least one job J_j such that $s_j < s^*_j$. Then, such a job J_j has $s^*_j > (\frac{1}{(\alpha-1)\beta})^{\frac{1}{\alpha}}$ which contradicts the fact that $\mathcal{S}^*$ is a regular schedule. We assume now that $L_{max} < L^*_{max}$. Then, there is at least one job J_j which is executed with different speeds in the two schedules. Let J_j be the job with the smallest index such that $s_j \neq s^*_j$. Obviously, $s_j > s^*_j$. Hence, J_{j-1} is critical in $\mathcal{S}^*$. But, by the way J_j was chosen, $L_{max} \geq L_{j-1} = L^*_{j-1} = L^*_{max}$, a contradiction. □

Based on this observation, we proceed to the description of our algorithm. In the first step, the algorithm assigns to every job J_j a speed s_j equal to $(\frac{1}{(\alpha-1)\beta})^{\frac{1}{\alpha}}$. In this way, we identify the value of the maximum lateness and the set of jobs executed with speed $(\frac{1}{(\alpha-1)\beta})^{\frac{1}{\alpha}}$ in the optimal schedule. This can be done by determining the highest-index critical job J_k in $\mathcal{S}_c$. All jobs $J_1, J_2, \ldots, J_k$ are executed with speed $(\frac{1}{(\alpha-1)\beta})^{\frac{1}{\alpha}}$ in $\mathcal{S}^*$. Moreover, all jobs with index greater than k have lateness strictly less than the maximum lateness of the optimal schedule. Therefore, we can decrease their speeds in order to reduce their energy consumption without affecting the maximum lateness of the schedule. This is done as follows: At the beginning, the algorithm has already assigned a speed to jobs $J_1, J_2, \ldots, J_k$. For each job J_j, $k+1 \leq j \leq n$, the algorithm defines a *candidate speed* of J_j, which we denote v_j. This speed is such that job J_j becomes critical given that J_k is critical and all jobs $J_{k+1}, J_{k+2}, \ldots, J_j$ are executed at the same speed. By Corollary 2.2, $v_j = \frac{1}{q_k - q_j} \sum_{i=k+1}^{j} w_i$. Then, among the candidate speeds, we choose the maximum one $v_{\max} = \max_j\{v_j\}$ and let J_ℓ be the job with the highest index, with $v_\ell = v_{max}$. We set the speed of jobs $J_{k+1}, J_{k+2}, \ldots, J_\ell$ equal to v_ℓ. Then, we set $k = \ell$ to be the highest index critical job in the current schedule and we proceed to the next step. The algorithm terminates when job n becomes critical. The complexity of the algorithm is $O(n^2)$, since each iteration of the while loop takes time at most $O(n)$. A pseudocode can be found in Algorithm 2.4.

Theorem 2.6. *Algorithm 2.4 is optimal for* $\mathbf{S}, \mathbf{1}||\mathbf{L_{max}} + \beta\mathbf{E}$.

Proof. In order to prove the theorem, we must show that Algorithm 2.4 always produces a schedule satisfying Lemma 2.5. We refer to such a schedule as *regular*. We will prove this by induction on the number of steps of the algorithm. At the end of each step, if J_ℓ is the last critical job in the current schedule, then the part of the schedule, from job J_1 up to job J_ℓ, is a regular one.

Initially, we consider the schedule produced just after the execution of line 3. It holds that all jobs are executed at a constant speed $s = (\frac{1}{(\alpha-1)\beta})^{\frac{1}{\alpha}}$ according to the EDD rule,

Algorithm 2.4

1: Order jobs according to the EDD order.
2: Assign to each job the speed $(\frac{1}{(\alpha-1)\beta})^{\frac{1}{\alpha}}$.
3: Let J_k be the highest-index critical job in the current schedule.
4: **while** $k < n$ **do**
5: **for** $j = k$ to n **do**
6: Compute v_j assuming that J_k and J_j are consecutive critical jobs.
7: Set the speed of jobs $J_k, J_{k+1}, \ldots, J_n$ equal to $v_{max} = \max_{k \leq j \leq n}\{v_j\}$.
8: Let J_ℓ be the highest-index critical job in the current schedule.
9: $k = \ell$

without any idle period between them. Let J_k be the last critical job in the current schedule. It is obvious that every non-critical job J_j, $1 \leq j \leq k$, has equal speed with the job that follows. Moreover, all the speeds of jobs $J_1, J_2, \ldots, J_k$ are equal, i.e. non-increasing, and it is obvious that the first job is executed with speed $s_1 = (\frac{1}{(\alpha-1)\beta})^{\frac{1}{\alpha}}$. Therefore, the initial schedule is regular.

Now assume that, up to step i, the schedule for jobs $J_1, J_2, \ldots, J_k$ is regular, where job J_k is the critical job with the highest index in the current schedule at the end of step i. Let $\ell > k$ be the highest-index critical job at the end of step $i+1$. We will show that, at the end of step $i+1$, the schedule for $J_1, J_2, \ldots, J_\ell$ is regular.

To begin with, it is clear that in the current schedule at the end of the step $i+1$, every job is executed at a constant speed, the jobs are executed according to the EDD order and there is no idle period between any jobs. Moreover, by construction, job J_ℓ is critical. Due to induction hypothesis, every non-critical job J_j, $1 \leq j \leq k$, has equal speed with the job J_{j+1} and the speeds of jobs $1, 2, \ldots, k$ are in non-increasing order. At the $i+1$-th step, all jobs $J_{k+1}, J_{k+2}, \ldots, J_n$ are assigned an equal speed which is less than the s_k since, otherwise, k wouldn't be the last critical job in the current schedule at the end of the i-th step. Therefore, in the current schedule at the end of the $i+1$-th step it holds that every non-critical job J_j, $1 \leq j \leq k$, has equal speed with the job J_{j+1} and the speeds of jobs $J_1, J_2, \ldots, J_\ell$ are in non-increasing order. Moreover, the speed of the first job does not change and remains equal to $(\frac{1}{(\alpha-1)\beta})^{\frac{1}{\alpha}}$. □

Online Algorithm for $\mathbf{S}, \mathbf{1}|\mathbf{r_j}|\mathbf{L_{max}} + \beta\mathbf{E}$

Let us, now, move to our online algorithm for $\mathbf{S}, \mathbf{1}|\mathbf{r_j}|\mathbf{L_{max}} + \beta\mathbf{E}$. We denote by $\mathcal{S}^*(\mathcal{J}, t)$ the optimal offline schedule of a set of jobs $\mathcal{J}$ with a common release date at time t. Subsequently, we give a description of our algorithm in Algorithm 2.5.

Algorithm 2.5

Let $\mathcal{R}_0$ be the set of jobs released at time $t_0 = 0$. In the phase 0, the jobs in $\mathcal{R}_0$ are scheduled according to $\mathcal{S}^*(\mathcal{R}_0, t_0)$. Let t_1 be the time at which the last job of $\mathcal{R}_0$ is finished and $\mathcal{R}_1$ be the set of jobs released during $(t_0, t_1]$. In the phase 1, the jobs in $\mathcal{R}_1$ are scheduled as in $\mathcal{S}^*(\mathcal{R}_1, t_1)$ and so on. In general, if t_i is the completion time of $\mathcal{S}^*(\mathcal{R}_{i-1}, t_{i-1})$, we denote $\mathcal{R}_i$ to be the set of jobs released during $(t_{i-1}, t_i]$. The jobs in $\mathcal{R}_i$ are scheduled by computing $\mathcal{S}^*(\mathcal{R}_i, t_i)$.

Figure 2.6: The structure of the schedule produced by the Algorithm 2.5.

Next, we analyze the competitive ratio of the algorithm.

Theorem 2.7. *Algorithm 2.5 is 2-competitive for the online version of the problem* $\mathbf{S}, \mathbf{1}|\mathbf{r_j}|\mathbf{L_{max}} + \beta\mathbf{E}$.

Proof. Assume that Algorithm 2.5 produces a schedule in $\ell + 1$ phases. Recall that the jobs of the i-th phase, i.e. the jobs in $\mathcal{R}_i$, are released during $(t_{i-1}, t_i]$ and scheduled as in $\mathcal{S}^*(\mathcal{R}_i, t_i)$. Let $L_{max,i} + \beta E_i$ be the cost of $\mathcal{S}^*(\mathcal{R}_i, t_i)$, where $L_{max,i}$ is the maximum lateness among the jobs in $\mathcal{R}_i$ and E_i is the energy consumed by the jobs of $\mathcal{R}_i$. The objective value of the algorithm's schedule is

$$SOL = \max_{0 \le i \le \ell} \{L_{max,i}\} + \beta \sum_{i=0}^{\ell} E_i \tag{2.19}$$

Now, we consider the optimal schedule. To lower bound the objective value OPT of an optimal schedule, we round down the release dates of the jobs; the release dates of the jobs in phase i, are rounded down to t_{i-1}. Let $\mathcal{S}_d^*$ and OPT_d be an optimal offline schedule for the rounded instance and its cost, respectively. Clearly, the optimal offline schedule for the initial instance is feasible for the rounded one. Thus, $OPT \ge OPT_d$.

To lower bound OPT_d we consider a scheduling problem with restricted assignments, i.e. a problem where each job can only be executed by a subset of the available processors. We denote by $\mathcal{S}_m^*$ and OPT_m an optimal offline schedule and its cost, respectively. The instance of this problem consists of $\ell + 1$ processors $P_0, P_1, \ldots, P_\ell$ and the set of jobs $\mathcal{J}$, where the release dates of the jobs are rounded down, as before. Jobs in $\mathcal{R}_0$ can only be assigned to the processor P_0 and every job in $\mathcal{R}_i$, $1 \le i \le \ell$, can only be executed by one of the processors P_0 or P_i. Moreover, it is required that all jobs in $\mathcal{R}_i$, $0 \le i \le \ell$, are executed by the same processor. Obviously, $OPT_d \ge OPT_m$ since $\mathcal{S}_d^*$ is a feasible schedule for the scheduling problem with restricted assignments.

Let us now describe an optimal offline schedule $\mathcal{S}_m^*$. Through a simple exchange argument, it can be shown that all the jobs in $\mathcal{R}_i$, $0 \leq i \leq \ell$, are executed by the processor P_i in an optimal schedule. In such a schedule, the jobs in $\mathcal{R}_i$, $1 \leq i \leq \ell$, are scheduled according to $\mathcal{S}^*(\mathcal{R}_i, t_{i-1})$, while the jobs in $\mathcal{R}_0$ are scheduled with respect to $\mathcal{S}^*(\mathcal{R}_0, t_0)$. Assume that the maximum lateness of the above schedule is attained by a job of the set $\mathcal{R}_k$, $0 \leq k \leq \ell$, executed by the processor P_k. So, $L_{max}^* = L_{max,k}^*$, where L_{max}^*, $L_{max,k}^*$ is the maximum lateness of the schedules $\mathcal{S}_m^*$, $\mathcal{S}^*(\mathcal{R}_i, t_{i-1})$, respectively. Let E_i^* be the energy consumption of the schedule $\mathcal{S}^*(\mathcal{R}_i, t_{i-1})$. Then,

$$OPT_m = L_{max,k}^* + \beta \sum_{i=0}^{\ell} E_i^* \tag{2.20}$$

By considering the schedules $\mathcal{S}^*(\mathcal{R}_i, t_{i-1})$ and $\mathcal{S}^*(\mathcal{R}_i, t_i)$, it can be easily shown that $L_{max,i}^* = L_{max,i} - (t_i - t_{i-1})$ and $E_i^* = E_i$. Hence, by (2.19) and (2.20) these imply that $OPT_m = SOL - (t_k - t_{k-1})$. Note that $t_k - t_{k-1}$ is the total processing time of the jobs in $\mathcal{R}_{k-1}$ in the schedule produced by Algorithm 2.5 which is equal to the total processing time of the jobs in $\mathcal{R}_{k-1}$ in $\mathcal{S}_m^*$. Recall also that the last job of each set $\mathcal{R}_i$ attains lateness at most $L_{max,i}$ in $\mathcal{S}_m^*$. Thus,

$$t_k - t_{k-1} \leq L_{max,k-1}^* \leq OPT_m$$

Therefore, $SOL \leq 2OPT$ and the Algorithm 2.5 is 2-competitive for the online version of the problem $\mathbf{S}, \mathbf{1}|\mathbf{r_j}|\mathbf{L_{max}} + \beta\mathbf{E}$. □

Chapter 3

Homogeneous Parallel Processors

In this chapter, we study energy aware scheduling problems on homogeneous parallel processors. The processors are homogeneous in the sense that they all obey the same speed-to-power function.

In Section 3.1, we begin with the energy minimization problem $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$ in which we allow preemptions and migrations of the jobs. For this problem, we propose two optimal polynomial time algorithms. The first algorithm is based on repeated maximum flow computations while the second one is based on a formulation of the problem as a convex cost flow problem.

Next, in Section 3.2, we study the non-preemptive non-migratory energy minimization problem $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{agrbl}|\mathbf{E}$ for agreeable instances and we present a $(2 - \frac{1}{m})^{\alpha-1}$-approximation algorithm.

3.1 Energy Minimization with Migrations and Preemptions

In this section, we consider the problem $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$ for which we propose two optimal polynomial time algorithms. The former one is based on a series of maximum flow computations while the latter one is based on a single minimum convex cost flow calculation.

In the problem $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$, we have to schedule a set of n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ on a set of m parallel processors $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$ so as to minimize the total energy consumption. Each job $J_j \in \mathcal{J}$ is specified by a work w_j, a release date r_j and a deadline d_j and it must be entirely executed during the interval $[r_j, d_j)$. We allow preemptions and migrations of the jobs, i.e. a job may be executed, suspended and resumed later from the point of suspension on the same or on another processor. However, we do not allow parallel execution of a job. That is, each job can be executed by at most one processor at each time.

3.1.1 Optimal Algorithm based on Maximum Flow

In the following, we present a maximum flow based algorithm for $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$. In order to establish this polynomial algorithm, we first formulate the problem as a convex program. This convex programming formulation gives a straightforward polynomial time algorithm for the problem because we can solve convex programs in polynomial time by applying the Ellipsoid algorithm. Next, we apply the KKT conditions to this convex program and we derive some necessary and sufficient conditions for optimality. Then, we define an optimal algorithm for the problem which always constructs a solution satisfying the KKT conditions. The algorithm is based on a series of repeated maximum flow computations on an appropriate graph.

We define the times $t_0 < t_1 < \ldots < t_\tau$ so that there is exactly one time t_k, $0 \leq k \leq \tau$ for every possible release date and deadline. Note that $\tau = O(n)$. Let $I_k = [t_{k-1}, t_k)$, for $1 \leq k \leq \tau$, and $\mathcal{I} = \{I_1, I_2, \ldots, I_\tau\}$. We denote by $|I_k|$ the length of the interval I_k, i.e. $|I_k| = t_k - t_{k-1}$. We say that the job $J_j \in \mathcal{J}$ is *active* during the interval $I_k \in \mathcal{I}$ if $I_k \subseteq [r_j, d_j)$. Let $\mathcal{A}(I_k)$ the set of the jobs which are active during I_k. Additionally, let $n_k = |\mathcal{A}(I_k)|$ be the number of the active jobs during I_k.

Next, we describe a problem which is a variation of our problem and we call it the *Work Assignment Problem* (or WAP). We have a set of n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$, a set of m parallel processors $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$ and a set of τ disjoint intervals $\mathcal{I} = \{I_1, I_2, \cdots, I_\tau\}$. Each job J_j is associated with an amount of work w_j. For a given interval $I_k \in \mathcal{I}$ and a job $J_j \in \mathcal{J}$ there are two cases: either the job J_j can be executed during I_k or it cannot. In the first case, we say that J_j is *active* during I_k. Following our existing definition, we denote by $\mathcal{A}(I_k)$ and n_k the set and the number of active jobs during I_k. During each interval $I_k \in \mathcal{I}$ there is a set $\mathcal{P}(I_k)$ of m_k available processors. Preemptions and migrations of jobs are allowed but no parallel execution of a job is permitted. Moreover, we are given a speed value v. Our objective is to find whether or not there is a feasible schedule that executes all jobs in $\mathcal{J}$ with constant speed v. Note that a schedule is feasible if and only if each job is entirely executed during its active intervals and it is executed by at most one among the available processor at each time.

Note that the WAP is a generalization of the multiprocessor feasibility scheduling problem $\mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|-$ where, given a set of jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ such that each job J_j has a processing time p_j, a release date r_j and a deadline d_j, and a set of identical parallel processors, we ask whether there exists a feasible preemptive and migratory schedule that executes each job between its release date and its deadline. The problem $\mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|-$ is almost the same as the WAP with the difference that, in the WAP, not all intervals have the same number of available processors. Note that each job has a fixed processing time in the WAP since the speed is part of the problem's instance. The WAP is polynomially solvable by applying a variant of an optimal algorithm for $\mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|-$ (see [1]).

Convex Programming Formulation and KKT Conditions

In order to derive a convex program for our problem, we first observe that, in any optimal schedule, every job $J_j \in \mathcal{J}$ is executed at a constant speed and this comes from the convexity of the speed-to-power function. So, we introduce a variable s_j and a variable

$p_{j,k}$, for each $J_j \in \mathcal{J}$ and for all I_k such that $J_j \in \mathcal{A}(I_k)$, to be the speed of job J_j and the total execution time of job J_j during the interval I_k, respectively. Then, we propose the following convex programming formulation for the problem $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$.

$$\min \sum_{J_j \in \mathcal{J}} w_j s_j^{\alpha-1} \tag{3.1}$$

$$\sum_{I_k:\, J_j \in \mathcal{A}(I_k)} p_{j,k} = \frac{w_j}{s_j} \qquad J_j \in \mathcal{J} \tag{3.2}$$

$$\sum_{J_j \in \mathcal{A}(I_k)} p_{j,k} \leq m \cdot |I_k| \qquad I_k \in \mathcal{I} \tag{3.3}$$

$$\sum_{J_j \in \mathcal{A}(I_k)} p_{j,k} \leq n_k \cdot |I_k| \qquad I_k \in \mathcal{I} \tag{3.4}$$

$$p_{j,k} \leq |I_k| \qquad I_k \in \mathcal{I},\ J_j \in \mathcal{A}(I_k) \tag{3.5}$$

$$p_{j,k} \geq 0 \qquad I_k \in \mathcal{I},\ J_j \in \mathcal{A}(I_k) \tag{3.6}$$

$$s_j \geq 0 \qquad J_j \in \mathcal{J} \tag{3.7}$$

Note that the total processing time and the total energy consumption of a job J_j executed with speed s_j is $\frac{w_j}{s_j}$ and $w_j s_j^{a-1}$, respectively. Thus, the term (3.1) is the total energy consumed foe all the jobs which is our objective function. The constraints (3.2) enforce that w_j units of work must be executed for each job J_j in total. The constraints (3.3) ensure that we use at most m processors for $|I_k|$ units of time during any interval $I_k \in \mathcal{I}$. Also, we can use at most $|\mathcal{A}(I_k)|$ processors operating for $|I_k|$ units of time during any interval $I_k \in \mathcal{I}$, otherwise we would have parallel execution of a job and this is expressed by the constraints (3.4). The constraints (3.5) prevent any job J_j from being executed for more than $|I_k|$ units of time during any interval $I_k \subseteq [r_j, d_j)$, otherwise we would have parallel execution of a job. The constraints (3.6) and (3.7) ensure the non-negativity of the variables $p_{j,k}$ and s_j, respectively, for every job and any possible interval during which the job is active.

The above mathematical program is indeed convex because the objective function and the first constraint are convex for $\alpha > 2$ while all the other constraints are linear. Since our problem can be written as a convex program, it can be solved in polynomial time by applying the Ellipsoid Algorithm. At this point, notice that once the speeds, i.e. the processing times, of the jobs are computed, by solving the convex program, a further step is needed in order to construct the optimal schedule. This step consists of solving the feasibility problem $\mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|-$.

Next, we apply the KKT conditions to the above convex program so as to obtain necessary and sufficient conditions that any schedule must satisfy in order to be optimal. The general form of the KKT conditions can be found in the Appendix A.

Lemma 3.1. *There is always an optimal schedule for the problem* $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$ *that satisfies the following properties:*

1. *Each job* J_j *is executed at a constant speed* s_j.
2. *For any interval* $I_k \in \mathcal{I}$, *we have that* $\sum_{J_j \in \mathcal{A}(I_k)} p_{j,k} = \min\{n_k, m\}|I_k|$.

3. *For any interval $I_k \in \mathcal{I}$ such that $n_k \leq m$, it holds that $p_{j,k} = |I_k|$ for every job $J_j \in \mathcal{A}(I_k)$.*

4. *For any interval $I_k \in \mathcal{I}$ such that $n_k > m$, it holds that*

   i. *All jobs $J_j \in \mathcal{A}(I_k)$ with $0 < p_{j,k} < |I_k|$ have equal speeds.*

   ii. *If a job $J_j \in \mathcal{A}(I_k)$ is not executed during I_k, i.e. $p_{j,k} = 0$, then $s_j \leq s_{j'}$ for any job $J_{j'} \in \mathcal{A}(I_k)$ with $p_{j',k} > 0$.*

   iii. *If a job $J_j \in \mathcal{A}(I_k)$ is executed during the whole interval I_k, i.e. $p_{j,k} = |I_j|$, then $s_j \geq s_{j'}$ for any job $J_{j'} \in \mathcal{A}(I_k)$ with $p_{j',k} < |I_j|$.*

Proof. The proofs of the properties 1, 2 and 3 of the lemma are omitted because thay can be easily proved by applying the definition of convexity and simple exchange arguments. Next, we focus on proving the property 4 based on the KKT conditions. In order to apply the KKT conditions, we need to associate with each constraint of the convex program a dual variable. Therefore, to each set of the constraints from (3.2) up to (3.6), we associate the dual variables ξ_j, λ_k, μ_k, $\pi_{j,k}$ and $\sigma_{j,k}$, respectively. Without loss of generality, we assume that $s_j > 0$ for each job $J_j \in \mathcal{J}$ in any feasible schedule. Therefore, by the complementary slackness conditions, it holds that the dual variables associated with the constraints (3.7) are equal to zero in any optimal solution.

By stationarity conditions, we have that

$$\begin{aligned}
&\nabla\left(\sum_{J_j\in\mathcal{J}} w_j s_j^{\alpha-1}\right) + \sum_{J_j\in\mathcal{J}} \xi_j \cdot \nabla\left(\frac{w_j}{s_j} - \sum_{I_k:\, J_j\in\mathcal{A}(I_k)} p_{j,k}\right) \\
+ \sum_{I_k\in\mathcal{I}} \lambda_k \nabla\left(\sum_{J_j\in\mathcal{A}(I_k)} p_{j,k} - m\cdot|I_k|\right) &+ \sum_{I_k\in\mathcal{I}} \mu_k \nabla\left(\sum_{J_j\in\mathcal{A}(I_k)} p_{j,k} - n_k\cdot|I_k|\right) \\
&+ \sum_{I_k\in\mathcal{I}}\sum_{J_j\in\mathcal{A}(I_k)} \pi_{j,k}\nabla(p_{j,k} - |I_k|) + \sum_{I_k\in\mathcal{I}}\sum_{J_j\in\mathcal{A}(I_k)} \sigma_{j,k}\nabla(-p_{j,k}) \;=\; 0 \Leftrightarrow
\end{aligned}$$

$$\begin{aligned}
\sum_{I_k\in\mathcal{I}}\sum_{J_j\in\mathcal{A}(I_k)} &\left(-\xi_j + \lambda_k + \mu_k + \pi_{j,k} - \sigma_{j,k}\right)\nabla p_{j,k} \\
&+ \sum_{J_j\in\mathcal{J}} \left((\alpha-1) w_j s_j^{\alpha-2} - \frac{\xi_j w_j}{s_j^2}\right)\nabla s_j \;=\; 0
\end{aligned}$$

We set the coefficients of the partial derivatives ∇s_j and $\nabla p_{j,k}$ equal to zero so as to satisfy the stationarity conditions. Thus, we get equivalently that

$$(\alpha-1)s_j^{\alpha} = \lambda_k + \mu_k + \pi_{j,k} - \sigma_{j,k} \qquad I_k \in \mathcal{I},\ J_j \in \mathcal{A}(I_k) \tag{3.8}$$

The complementary slackness are stated as follows.

$$\lambda_k \cdot \left(\sum_{J_j \in \mathcal{A}(I_k)} p_{j,k} - m \cdot |I_k| \right) = 0 \qquad I_k \in \mathcal{I} \tag{3.9}$$

$$\mu_k \cdot \left(\sum_{J_j \in \mathcal{A}(I_k)} p_{j,k} - n_k \cdot |I_k| \right) = 0 \qquad I_k \in \mathcal{I} \tag{3.10}$$

$$\pi_{j,k} \cdot (p_{j,k} - |I_k|) = 0 \qquad I_k \in \mathcal{I},\ J_j \in \mathcal{A}(I_k) \tag{3.11}$$

$$\sigma_{j,k} \cdot (-p_{j,k}) = 0 \qquad I_k \in \mathcal{I},\ J_j \in \mathcal{A}(I_k) \tag{3.12}$$

We consider an interval $I_k \in \mathcal{I}$ such that $n_k > m$. Because of the property 2 and (3.10), we have that $\mu_k = 0$. Next, we consider the following cases for the execution time of any job $J_j \in \mathcal{A}(I_k)$ during I_k:

- $0 < p_{j,k} < |I_k|$
  Complementary slackness conditions (3.11) and (3.12) imply that $\pi_{j,k} = \sigma_{j,k} = 0$. As a result, (3.8) can be written as

  $$(\alpha - 1)s_j^\alpha = \lambda_k. \tag{3.13}$$

  The variable λ_k is specific for the interval I_k and, thus, all such jobs have the same speed throughout the whole schedule and the property $4(i)$ is valid.

- $p_{j,k} = 0$
  This means, by (3.11), that $\pi_{j,k} = 0$ and (3.8) is expressed as $(\alpha-1)s_j^\alpha = \lambda_k - \sigma_{j,k}$. Thus, since $\sigma_{j,k} \geq 0$, we get that

  $$(\alpha - 1)s_j^\alpha \leq \lambda_k. \tag{3.14}$$

- $p_{j,k} = |I_k|$
  In this case, by (3.12), we get that $\sigma_{j,k} = 0$. So, (3.8) becomes $(\alpha-1)s_j^\alpha = \lambda_k + \pi_{j,k}$. Because of dual feasibility conditions, $\pi_{j,k} \geq 0$. Hence, all jobs of this kind have

  $$(\alpha - 1)s_j^\alpha \geq \lambda_k. \tag{3.15}$$

By Equations (3.13), (3.14) and (3.15), we get the properties $4(ii)$ and $4(iii)$. □

Given a solution of the convex program that satisfies the KKT conditions, we derived some relations between the primal variables. Based on them, we defined some structural properties of any optimal schedule. These properties are necessary for optimality and we show that they are also sufficient because all schedules that satisfy these properties attain equal energy consumptions.

Lemma 3.2. *The conditions of Lemma 3.1 are also sufficient for optimality.*

Proof. Assume for the sake of contradiction that there is a schedule $\mathcal{S}$, that satisfies the properties of Lemma 3.1, which is not optimal and let $\mathcal{S}^*$ be an optimal schedule that also satisfies the properties (by Lemma 3.1 we know that the schedule $\mathcal{S}^*$ always exists). We

denote E, s_j and $p_{j,k}$ the energy consumption, the speed of job J_j and the total execution time of job J_j during the interval I_k, respectively, in schedule $\mathcal{S}$. Let E^*, s_j^* and $p_{j,k}^*$ be the corresponding values for the schedule $\mathcal{S}^*$. Let $\mathcal{J}'$ be the set of jobs J_j with $s_j > s_j^*$. Clearly, there is at least one job J_j such that $s_j > s_j^*$, otherwise $\mathcal{S}$ would not consume more energy than $\mathcal{S}^*$. So, $\mathcal{J}' \neq \emptyset$. By definition of $\mathcal{J}'$,

$$\sum_{J_j \in \mathcal{J}'} \sum_{I_k : J_j \in \mathcal{A}(I_k)} p_{j,k} < \sum_{J_j \in \mathcal{J}'} \sum_{I_k : J_j \in \mathcal{A}(I_k)} p_{j,k}^*.$$

Hence, there is at least one interval I_k such that

$$\sum_{J_j \in \mathcal{J}' \cup \mathcal{A}(I_k)} p_{j,k} < \sum_{J_j \in \mathcal{J}' \cup \mathcal{A}(I_k)} p_{j,k}^*.$$

If $n_k \leq m$, then there is at least one job $J_{j'}$ such that $p_{j',k} < p_{j',k}^*$. Due to the property 3 of Lemma 3.1, it should hold that $p_{j',k} = p_{j',k}^* = |I_k|$ which is a contradiction. So, it must be the case that $n_k > m$. Then, the last equation gives that $p_{j,k} < p_{j,k}^*$ for some job $J_j \in \mathcal{A}(I_k)$. Thus, $p_{j,k} < |I_k|$ and $p_{j,k}^* > 0$. Both schedules must have equal sum of processing times $\sum_{J_{j'} \in \mathcal{A}(I_k)} p_{j',k}$ during the interval I_k. So, there must be a job $J_{j'} \notin \mathcal{J}'$ such that $p_{j',k} > p_{j',k}^*$. Therefore, $p_{j',k} > 0$ and $p_{j',k}^* < |I_k|$. We conclude that $s_{j'} \geq s_j > s_j^* \geq s_{j'}^*$, which contradicts the fact that $J_{j'} \notin \mathcal{J}'$. □

Notice that the properties of Lemma 3.1 do not explain how to find an optimal schedule. The basic reason is that they do not determine the exact speed value of each job. Moreover, they do not specify exactly the structure of the optimal schedule. That is, they do not specify which job is executed by each processor at each time.

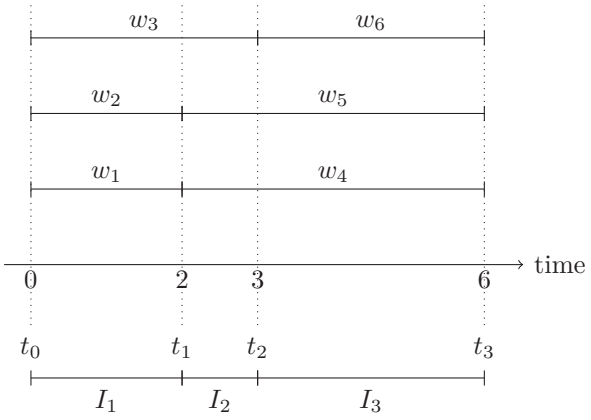
Optimal Combinatorial Algorithm

Next, we propose an optimal combinatorial algorithm for the problem $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$ that always constructs a schedule satisfying the properties of Lemma 3.1 which, as we have already showed, are necessary and sufficient for optimality.

Our algorithm is based on the notion of *critical jobs* defined below. Initially, the algorithm conjectures that all jobs are executed at the same speed in the optimal schedule and it assigns to all of them a speed which is an upper bound on the speed that any job has in the optimal schedule. The key idea is to continuously decrease the speeds of the jobs step by step. At each step, it assigns a speed to the critical jobs which are ignored in the subsequent steps and it goes on reducing the speeds of the remaining jobs. At the end of the last step, every job has been assigned a speed. Critical jobs are recognized by finding a minimum (s,t)-cut in an appropriate graph as we describe in the following. Once the algorithm has computed a speed, i.e. a processing time, for each job, it constructs a feasible schedule by applying an optimal algorithm for $\mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|-$.

At a given step, the algorithm performs a binary search in order to reduce the speeds of the jobs. The binary search is performed by solving repeatedly different instances of the WAP. Each instance of the WAP is solved by a maximum flow computation. Specifically, given an instance $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v >$ of the WAP, the algorithm constructs a directed graph G as follows. There is one node for each job $J_j \in \mathcal{J}$, one node for each interval $I_k \in \mathcal{I}$, a

source node s and a destination node t. The algorithm introduces an arc (s, J_j), for each $J_j \in \mathcal{J}$, with capacity $\frac{w_j}{v}$, an arc (J_j, I_k) with capacity $|I_k|$, for each couple of job J_j and interval I_k such that $J_j \in \mathcal{A}(I_k)$, and an arc (I_k, t) with capacity $m_k|I_k|$ for each interval $I_k \in \mathcal{I}$. We say that this is the *corresponding graph* of $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v >$. The algorithm decides if an instance $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v >$ of the WAP is feasible by computing a maximum (s, t)-flow on its corresponding graph G, based on the following lemma.



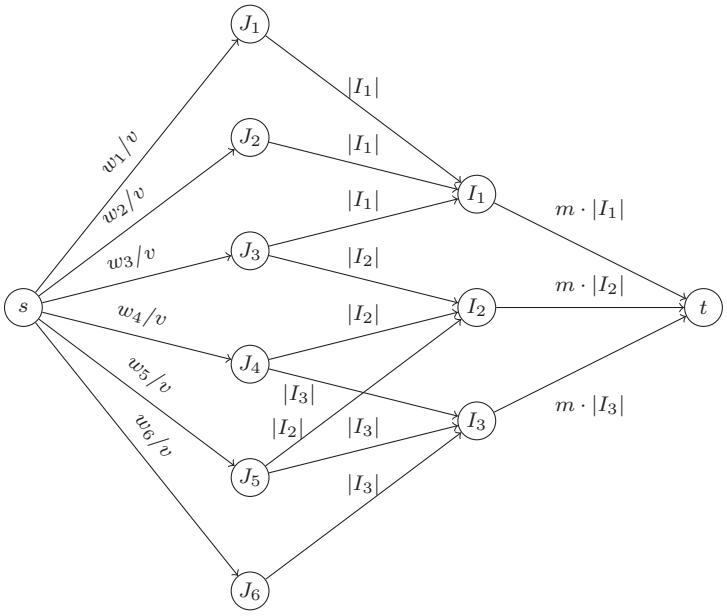


Figure 3.1: An instance of the WAP such that m processors are available during each interval I_k and its corresponding graph.

Theorem 3.1. *There exists a feasible schedule for an instance $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v >$ of the WAP if and only if there exists a feasible (s,t)-flow of value $\sum_{J_j \in \mathcal{J}} \frac{w_j}{v}$ in the corresponding graph G.*

We are ready to introduce the notion of criticality for feasible instances of the WAP. Given a feasible instance for the WAP, we say that job J_c is *critical* if, for any feasible schedule $\mathcal{S}$ and for each interval I_k such that $J_j \in \mathcal{A}(I_k)$, either $p_{c,k} = |I_k|$ or $\sum_{J_j \in \mathcal{A}(I_k)} p_{j,k} = m_k |I_k|$, where $p_{j,k}$ is the total amount of time that the job $J_j \in \mathcal{J}$ is processed during the interval I_k in $\mathcal{S}$. Moreover, we say that an instance $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v >$ of the WAP is critical if v is the minimum speed so that the set of jobs $\mathcal{J}$ can be feasibly executed during the intervals in $\mathcal{I}$. We refer to this speed v as the critical speed of $\mathcal{J}$, $\mathcal{P}$ and $\mathcal{I}$.

Based on the Theorem 3.1, we extend the notion of criticality. Let us consider a feasible instance $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v >$ of the WAP and let $G = (V, A)$ be its corresponding graph. Given an arc $e \in A$ and a feasible (s,t)-flow $\mathcal{F}$ of G, we say that the arc e is saturated by $\mathcal{F}$ if the amount of flow that crosses the arc e according to $\mathcal{F}$ is equal to the capacity of e. Additionally, we say that a path p of G is saturated by $\mathcal{F}$ if there exists at least one arc e in p which is saturated. Then, a job $J_c \in \mathcal{J}$ is critical if and only if, for any maximum (s,t)-flow $\mathcal{F}$ in G, either the arc (J_c, I_k) or the arc (I_k, t) is saturated, for each path J_c, I_k, t, i.e. for every I_k such that $J_c \in \mathcal{A}(I_k)$, according to $\mathcal{F}$. In other words, J_c is critical if every path J_c, I_k, t is saturated by any maximum (s,t)-flow $\mathcal{F}$.

In order to continue our analysis, we need the following lemma which relates, in a sense, the notions of *critical job* and *critical instance*.

Lemma 3.3. *If $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v >$ is a critical instance of WAP, then there is at least one critical job $J_j \in \mathcal{J}$.*

Proof. Let G be the corresponding graph of $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v >$. Since the instance $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v >$ is critical, there exists a minimum (s,t)-cut $\mathcal{C}$ in G that contains either an arc (J_j, I_k), for some $J_j \in \mathcal{J}$ and $I_k \in \mathcal{I}$, or an arc (I_k, t), for some $I_k \in \mathcal{I}$. If this was not the case, the only minimum (s,t)-cut would be the one with all the arcs (s, J_j). This means that we could reduce the speed v to $v - \epsilon$, for an infinitesimal quantity $\epsilon > 0$, and the instance $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v - \epsilon >$ would admit a feasible flow equal to $\sum_{J_j \in \mathcal{J}} \frac{w_j}{v - \epsilon}$ which contradicts the criticality of $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v >$.

Now, there must be at least one arc (s, J_c) that does not belong to $\mathcal{C}$, which is a minimum (s,t)-cut containing at least one of the arcs (J_j, I_k) or (I_k, t). If all arcs (s, J_j) were included in $\mathcal{C}$, then $\mathcal{C}$ would have greater capacity than the (s,t)-cut that contains just all the arcs (s, J_j) in contradiction with the fact that $\mathcal{C}$ is a minimum (s,t)-cut. Based on the definition of an (s,t)-cut, we conclude that all paths J_c, I_k, t must have an arc that belongs in $\mathcal{C}$ so that if we remove the arcs of $\mathcal{C}$, the nodes s and t become disconnected. Hence, the job J_c is critical. □

Note that the instance $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v - \epsilon >$ is not feasible if $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v >$ is critical. Up to now, the notion of a critical job has been defined only in the context of feasible instances. We extend this notion for unfeasible instances as follows. In an unfeasible instance $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v - \epsilon >$, a job J_j is called critical if every path J_j, I_k, t is saturated by any maximum (s,t)-flow in the corresponding graph G'.

Let $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v >$ be a critical instance of the WAP and let G be its corresponding graph. Next, we propose a way for identifying the critical jobs of $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v >$ using the graph G' that corresponds to the instance $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v - \epsilon >$, for some sufficiently small constant $\epsilon > 0$ based on Lemmas 3.4 and 3.5 below. The value of ϵ is such that the two instances have exactly the same set of critical jobs. Moreover, the critical jobs of $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v - \epsilon >$ can be found by computing a minimum (s, t)-cut in the graph that corresponds to $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v - \epsilon >$.

Lemma 3.4. *Given a critical instance $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v >$ of the WAP, there exists a sufficiently small constant $\epsilon > 0$ such that the unfeasible instance $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v - \epsilon >$ and $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v >$ have exactly the same critical jobs. The same holds for any other unfeasible instance $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v - \epsilon' >$ such that $0 < \epsilon' \leq \epsilon$.*

Proof. Since $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v >$ is a critical instance, because of Lemma 3.3, it must contain at least one critical job.

If all the jobs of the instance are critical, then, in the graph G that corresponds to $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v >$, there is a minimum (s, t)-cut $\mathcal{C}$ that contains exactly one arc of every path J_j, I_k, t, $I_k \in \mathcal{I}$ and $J_j \in \mathcal{A}(I_k)$. Clearly, $\mathcal{C}$ is a minimum (s, t)-cut for the graph G' that corresponds to $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v - \epsilon >$ for any $\epsilon > 0$, because all the arcs (s, J_j), $J_j \in \mathcal{J}$, have greater capacity in G' than in G, while all the other arcs have equal capacities in the two graphs. Hence, for any job $J_j \in \mathcal{J}$, either the arc (J_j, I_k) or the arc (I_k, t) is saturated by any maximum (s, t)-flow in G', for all $I_k \in \mathcal{I}$ such that $J_j \in \mathcal{A}(I_k)$. That is, all jobs are critical in G' as well and the lemma is true.

Now, assume that there is at least one non-critical job. Consider a non-critical job J_j. We know that there must be at least one maximum (s, t)-flow $\mathcal{F}$ in G such that at least one path J_j, I_k, t is not saturated by $\mathcal{F}$, for some $I_k \in \mathcal{I}$ such that $J_j \in \mathcal{A}(I_k)$. Consider such a path J_j, I_k, t. Since the path is not saturated, we have that $c_{(J_j, I_k)} - f_{(J_j, I_k)} > 0$ and $c_{(I_k, t)} - f_{(I_k, t)} > 0$, where c_e is the capacity of the arc e and f_e is the amount of flow that passes through e according to $\mathcal{F}$, respectively. Then, we set

$$\eta_j = \min\{c_{(J_j, I_k)} - f_{(J_j, I_k)}, c_{(I_k, t)} - f_{(I_k, t)}\}$$

The intuition behind the value η_j is the following. Assume that we increase the capacity of the arc (s, J_j) while keeping the same capacities for the remaining arcs. If this increase is less than η_j, then there is a maximum (s, t)-flow $\mathcal{F}'$ in the new graph such that neither the arc (J_j, I_k), nor the arc (I_k, t) are saturated by $\mathcal{F}'$. The maximum (s, t)-flow $\mathcal{F}'$ in the new graph can be easily obtained from the maximum (s, t)-flow $\mathcal{F}$ in G.

For every non-critical job J_j, we fix a positive value η_j as we described in the previous paragraph. Note that we do not want to compute such a value but we only care for its existence. Let η_{min} be the minimum value η_j, among all the non-critical jobs. From the instance $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v >$, we obtain an unfeasible instance $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v - \epsilon >$ as follows. We pick an ϵ such that the total increase of the capacities of the all the arcs from the source node to the job nodes is less than η_{min}. In other words, the value ϵ must satisfy the following inequality

$$\sum_{J_j \in \mathcal{J}} \frac{w_j}{v - \epsilon} < \sum_{J_j \in \mathcal{J}} \frac{w_j}{v} + \eta_{min}$$

Let us, now, explain why the two instances have the same critical jobs. Initially, we will show that if a job is non-critical in G, then it remains non-critical in G'. By the way we picked ϵ, for any non-critical job J_j in G, there is always a maximum (s,t)-flow such that some path from J_j to t is not saturated in G'. Therefore, each non-critical job in G, remains a non-critical job in G'.

Next, consider a critical job J_j of $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v >$. By construction, the arc (s, J_j) has greater capacities in G' than in G and all the arcs (J_j, I_k) and (I_k, t), $J_j \in \mathcal{A}(I_k)$, have equal capacities in the two graphs. We conclude that (s, J_j) cannot belong to any minimum (s,t)-cut in G'. Thus, every path J_j, I_k, t is saturated by any maximum (s,t)-flow in G'. Therefore, if a job is critical in G, then it is critical in G' as well. □

The following lemma is a direct consequence of the definition of criticality.

Lemma 3.5. *Assume that $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v >$ is a critical instance for the WAP and let G' be the graph that corresponds to the instance $< \mathcal{J}, \mathcal{P}, \mathcal{I}, v - \epsilon >$, for any sufficiently small constant $\epsilon > 0$ in accordance with the Lemma 3.4. Then, any minimum (s,t)-cut $\mathcal{C}'$ of G' contains exactly:*

i. one arc of every path J_j, I_k, t for any critical job J_j,

ii. the arc (s, J_j) for each non-critical job J_j.

We are now ready to give a high level description of our algorithm. Initially, we will assume that the optimal schedule consumes an unbounded amount of energy and we assume that all jobs are executed with the same speed s_{UB}. This speed is such that there exists a feasible schedule that executes all jobs with the same speed. Then, we decrease the speed of all jobs up to a point where no further reduction is possible so as to obtain a feasible schedule. At this point, all jobs are assumed to be executed with the same speed, which is critical, and there is at least one job that cannot be executed with speed less than this, in any feasible schedule. The jobs that cannot be executed with speed less than the critical one form the current set of critical jobs. So, the critical job(s) is (are) assigned the critical speed and is (are) ignored after this point. That is, in what follows, the algorithm considers the subproblem in which some jobs are omitted (critical jobs), because they are already assigned the lowest speed possible (critical speed) so that they can be feasibly executed, and there are less than m processors during some intervals because these processors are dedicated to the omitted jobs.

In detail, the algorithm consists of a number of steps, where at each step a binary search is performed in order to determine the minimum speed so as to obtain a feasible schedule for the remaining jobs, i.e. the critical speed. We denote s_{crit} the critical speed and $\mathcal{J}_{crit}$ the set of critical jobs at a given step. In order to determine s_{crit} and $\mathcal{J}_{crit}$, we perform a binary search assuming that all the remaining jobs are executed with the same speed. Due to the convexity of the speed-to-power function, we know that each job J_j cannot be executed with speed less than its density $\delta_j = \frac{w_j}{d_j - r_j}$ in any optimal schedule. Therefore, given a set of jobs $\mathcal{J}$, we know that there does not exist an optimal schedule that executes all jobs with a speed $s < \max_{J_j \in \mathcal{J}}\{\delta_j\}$. Also, observe that if all jobs have speed $s = \max_{I_k \in \mathcal{I}}\{\frac{1}{|I_k|}\sum_{J_j \in \mathcal{A}(I_k)} w_j\}$, then we can construct a feasible schedule. These bounds define the search space of the binary search performed in the initial step. In the

next step the critical speed of the previous step is an upper bound on the speed of all remaining jobs and a lower bound is the maximum density among them. We use these updated bounds to perform the binary search of the current step and we go on like that. Algorithm 3.1 does what we have already described.

Algorithm 3.1

1: $s_{UB} = \max_k\{\frac{1}{|I_k|}\sum_{J_j \in \mathcal{A}(I_k)} w_j\}$, $s_{LB} = \max_{J_j \in \mathcal{J}}\{\delta_j\}$
2: **while** $\mathcal{J} \neq \emptyset$ **do**
3: Find the minimum speed s_{crit} so that the instance $< \mathcal{J}, \mathcal{P}, \mathcal{I}, s_{crit} >$ of the WAP is feasible, using binary search in the interval $[s_{LB}, s_{UB}]$ with repeated maximum flow computations.
4: Pick a sufficiently small $\epsilon > 0$.
5: Determine the set of critical jobs $\mathcal{J}_{crit}$ by computing a minimum (s,t)-cut in the graph G' that corresponds to the instance $< \mathcal{J}, \mathcal{P}, \mathcal{I}, s_{crit} - \epsilon >$.
6: For each $J_j \in \mathcal{J}_{crit}$, set $s_j = s_{crit}$.
7: $\mathcal{J} = \mathcal{J} \backslash \mathcal{J}_{crit}$.
8: Update the set of available processors for each interval $I_k \in \mathcal{I}$.
9: $s_{UB} = s_{crit}$, $s_{LB} = \max_{J_j \in \mathcal{J}}\{\delta_j\}$
10: Apply an optimal algorithm for $P|r_i, d_i, pmtn|-$ to schedule the jobs, where each job J_j has processing time w_j/s_j.

Algorithm 3.1 produces an optimal schedule, and this holds because any schedule constructed by the algorithm satisfies the properties of Lemma 3.1.

Theorem 3.2. *Algorithm 3.1 produces an optimal schedule.*

Proof. First of all, it is obvious that each job is executed with because every job is assigned exactly one speed in one step of the algorithm and the Property 1 of Lemma 3.1 is true.

Before proving the remaining properties, we need some definitions. Recall that, at each step of the algorithm, the critical jobs are assigned a speed, some processors during some intervals are dedicated to these jobs and they are ignored in subsequent steps. Consider the i-th step of the algorithm. At the beginning of the step, the remaining jobs $\mathcal{J}^{(i)}$, processors $\mathcal{P}^{(i)}$ and available intervals $\mathcal{I}^{(i)}$ form the new instance of the WAP for which the critical speed and jobs have to be determined. We denote $G^{(i)}$ the graph that corresponds to the instance $< \mathcal{J}^{(i)}, \mathcal{P}^{(i)}, \mathcal{I}^{(i)}, v >$ of the WAP, where the speed v varies between $s^{(i)}_{UB}$ and $s^{(i)}_{LB}$, i.e. the bounds of the step.

Assume for contradiction that the Property 2 does not hold in the algorithm's schedule. Then, there must be an interval $I_k \in \mathcal{I}$ during which $\sum_{J_j \in \mathcal{A}(I_k)} p_{j,k} < \min\{n_k, m\}|I_k|$, i.e. we can decrease the speed of some job and still get a feasible schedule. Note that it cannot be the case that $\sum_{J_j \in \mathcal{A}(I_k)} p_{j,k} > \min\{n_k, m\}|I_k|$ because Algorithm 3.1 assigns speeds only if there exists a feasible schedule with respect to these speeds. So, there must be a job $J_c \in \mathcal{A}(I_k)$ such that $p_{c,k} < |I_k|$ and there is an idle period during I_k such that J_c is not executed. Suppose that J_c became critical during the i-th step. Then, in the graph $G^{(i)}$, since J_c is a critical job, either the arc (J_c, I_k) or the arc (I_k, t)

belongs to a minimum (s,t)-cut and as a result, for any maximum flow in $G^{(i)}$, either $f_{(J_c,I_k)} = |I_k|$ or $f_{(I_k,t)} = m_k^{(i)}|I_k|$ where $m_k^{(i)}$ is the number of available processors during I_k at the beginning of the i-th step. Hence, we have a contradiction on the fact that $\sum_{J_j \in \mathcal{A}(I_k)} p_{j,k} < \min\{n_k, m\}|I_k|$ and $p_{c,k} < |I_k|$.

For the Property 3, we claim that, for an interval I_k with $n_k \leq m$, if a job $J_c \in \mathcal{A}(I_k)$ becomes critical at the i-th step, then the arc (J_c, I_k) becomes saturated by any maximum (s,t)-flow in $G^{(i)}$. If this was not the case, then there would be a maximum (s,t)-flow in $G^{(i)}$ such that $f_{(J_c,I_k)} < |I_k|$. Also, it should hold that $f_{(J_j,I_k)} \leq |I_k|$ for any other job $J_j \in \mathcal{J}^{(i)} \cup \mathcal{A}(I_k)$. Hence, $f_{(I_k,t)} < n_k^{(i)}|I_k| \leq m_k^{(i)}|I_k|$, where $n_k^{(i)} = |\mathcal{J}^{(i)}|$. So, neither the arc (J_c, I_k) nor the arc (I_k, t) would be saturated contradicting the criticality of J_c. Therefore, the total execution time of J_c during I_k is $|I_k|$.

Next we prove the Property 4. Initially, consider two jobs J_j and $J_{j'}$, active during an interval I_k, such that $0 < p_{j,k} < |I_k|$ and $0 < p_{j',k} < |I_k|$. We will show that the jobs are assigned equal speeds by the algorithm. For this, it suffices to show that they are assigned a speed at the end of the same step. So, assume for contradiction that J_j becomes critical before $J_{j'}$, at the end of the i-th step. Then, either the arc (J_j, I_k) or the arc (I_k, t) belongs to a minimum (s,t)-cut $\mathcal{C}$ in $G^{(i)}$. Since $0 < p_{j,k} < |I_k|$, we know that there exists a maximum (s,t)-flow in $G^{(i)}$ such that $0 < f_{(J_j,I_k)} < |I_k|$. Thus, it is the arc (I_k, t) that belongs in $\mathcal{C}$. Consequently, in $G^{(i)}$, the edge (I_k, t) is saturated by any maximum (s,t)-flow, and as a result, all the processors during the interval I_k are dedicated to the execution of some tasks at the end of the i-th step. Hence, $J_{j'}$ cannot be assigned a speed at a step later than the i-th and we have a contradiction. That is, Property *4(i)* is true.

For the Property *4(ii)*, consider the case where $p_{j,k} = 0$ for a job J_j during an interval $I_k \subseteq [r_j, d_j)$ and assume that J_j becomes critical at the i-th step. Then, either (I_k, t) does not appear in $G^{(i)}$, that is no processors are available during I_k, or (I_k, t) belongs to a minimum (s,t)-cut of $G^{(i)}$. If none of these was true, then all the arcs $(J_{j'}, I_k)$ would belong to a minimum (s,t)-cut, for all $J_{j'} \in \mathcal{A}(I_k)$ that appear in $G^{(i)}$. So, the arc (J_j, I_k) would be saturated by any maximum (s,t)-flow and we have a contradiction, since the fact that $p_{j,k} = 0$ implies that there exists a maximum (s,t)-flow with $f_{(J_j,I_k)} = 0$. In both cases, that is if I_k does not appear in $G^{(i)}$ or (I_k, t) belongs to a minimum (s,t)-cut of $G^{(i)}$, no job executed during I_k will be assigned a speed after the i-th step. Hence, all jobs $J_{j'}$ with $p_{j',k} > 0$ do not have lower speed than J_j.

Next, let J_j be a job with $p_{j,k} = |I_k|$ and assume that it is assigned a speed at the i-th step. As we have already shown, this cannot happen after a step where a job $J_{j'}$ with $0 < p_{j',k} < |I_k|$ is assigned a speed because after such a step, the interval I_k is no longer considered. Also, as we showed in the previous paragraph, J_j does not become critical after a job $J_{j'}$ with $p_{j',k} = 0$. The Property *4(iii)* follows.

Finally, because of Lemmas 3.4 and 3.5, Algorithm 3.1 correctly identifies the critical jobs at each step of the algorithm. The theorem follows. □

We turn, now, our attention to the complexity of the algorithm. Because of Lemma 3.3 at least one job (all critical ones) is scheduled at each step of the algorithm. Therefore, there will be at most n steps. Assume that U is the range of all possible values of speeds divided by our desired accuracy. Then, the binary search needs to check $O(\log U)$ values

of speed to determine the next critical speed at one step. That is, BAL performs $O(\log U)$ maximum flow calculations at each step. Thus, the overall complexity of our algorithm is $O(nf(n)\log U)$ where $f(|V|)$ is the complexity of computing a maximum flow in a graph with $|V|$ vertices.

3.1.2 Optimal Algorithm based on Convex Cost Flow

In the following, we present a polynomial time algorithm for $\mathbf{S},\mathbf{P}|\mathbf{r_j},\mathbf{d_j},\mathbf{mgtn}|\mathbf{E}$ by formulating it as a convex cost flow problem. This formulation lies on the fact that there is always an optimal schedule for $\mathbf{S},\mathbf{P}|\mathbf{r_j},\mathbf{d_j},\mathbf{mgtn}|\mathbf{E}$ such that each job is executed at a constant speed. A convex cost flow computation allows us to get the optimal speed s_j for every job $J_j \in \mathcal{J}$, and thus its total execution time $p_j = \frac{w_j}{s_j}$. Then, given the execution times of the jobs, the algorithm constructs a feasible schedule by applying a polynomial-time algorithm for the feasibility scheduling problem $\mathbf{P}|\mathbf{r_j},\mathbf{d_j},\mathbf{mgtn}|-$.

Recall that, in the feasibility scheduling problem $\mathbf{P}|\mathbf{r_j},\mathbf{d_j},\mathbf{mgtn}|-$, we are given a set of n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ and a set of m identical processors $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$. Each job $J_j \in \mathcal{J}$ is characterized by a processing time p_j, a release date r_j and a deadline d_j. The objective is to construct a schedule such that every job $J_j \in \mathcal{J}$ is processed for p_j units of time during the interval $[r_j, d_j)$ or decide that such a schedule does not exist. An optimal polynomial algorithm for $\mathbf{P}|\mathbf{r_j},\mathbf{d_j},\mathbf{mgtn}|-$ can be found in [1].

Given an instance of $\mathbf{S},\mathbf{P}|\mathbf{r_j},\mathbf{d_j},\mathbf{mgtn}|\mathbf{E}$, we consider that the time is partitioned into intervals defined by the release dates and the deadlines of jobs. That is, we define the time points $t_0, t_1, \ldots, t_\tau$, in increasing order, where each t_k, $0 \leq k \leq \tau$, corresponds to either a release date or a deadline, so that, for each release date and deadline of any job, there is a corresponding t_k. Then, we define the intervals $I_k = [t_{k-1}, t_k)$, for $1 \leq k \leq \tau$, and we denote by $|I_k|$ the length of I_k. We call a job J_j *active* in a given interval I_k, if $I_k \subseteq [r_j, d_j)$. The set of active jobs during the interval I_k is denoted by $\mathcal{A}(I_k)$. Moreover, let n_k be the number of the jobs which are active during I_k.

In order to establish a convex cost flow formulation (see Appendix C for the definition of the convex cost flow problem) for $\mathbf{S},\mathbf{P}|\mathbf{r_j},\mathbf{d_j},\mathbf{mgtn}|\mathbf{E}$, we construct a directed graph $G = (V, A)$, where V is the set of nodes and A the set of the arcs. In the graph $G = (V, A)$, we introduce a source node s, a destination node t, a node for each job $J_j \in \mathcal{J}$, and a node for each interval $I_k \in \mathcal{I}$. For each $J_j \in \mathcal{J}$, we add an arc (s, J_j) of infinite capacity and, for each $I_k \in \mathcal{I}$, we add an arc (I_k, t) with capacity equal to $m|I_k|$. If the job $J_j \in \mathcal{J}$ is active during the interval $I_k \in I$, i.e. $J_j \in \mathcal{A}(I_k)$, then we introduce an arc (J_j, I_k) with capacity $|I_k|$.

To each arc $e \in A$, we associate a convex cost function $g_e(x)$ which specifies the cost incurred if x units of flow cross the arc e. The arcs have the following cost functions:

- $g_{(s,J_j)}(x) = \frac{w_j^\alpha}{x^{\alpha-1}}$, for all $J_j \in \mathcal{J}$,
- $g_{(J_j,I_k)}(x) = 0$, for all $I_k \in \mathcal{I}$ and $J_j \in \mathcal{A}(I_k)$, and
- $g_{(I_k,t)}(x) = 0$, for all $I_k \in \mathcal{I}$.

Let $\mathcal{F}$ be a feasible (s,t)-flow in the graph G. We denote by f_e the amount of flow that crosses the arc $e \in A$ according to $\mathcal{F}$. Any feasible (s,t)-flow for the graph G

corresponds to a feasible schedule for $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$. Specifically, $f_{(s,J_j)}$ corresponds to the processing time of job J_j. In this case, $\frac{w_j}{f_{(s,J_j)}}$ is the speed of J_j and $\frac{w_j^\alpha}{f_{(s,J_j)}^{\alpha-1}}$ is the energy consumed for the execution of J_j. Furthermore, the flow passing through the edge (J_j, I_k) represents the amount of time that the job J_j is executed during the interval I_k. In the same vein, $f_{(I_k,t)}$ corresponds to the total execution time of all the jobs during the interval I_k. Hence, the total flow that leaves the source node and arrives to the destination node corresponds to the total execution time of all jobs. By Lemma 3.1, we get the following corollary which specifies the total execution time of all jobs in an optimal schedule for $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$.

Corollary 3.1. *In an optimal schedule for* $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$ *where each job* $J_j \in \mathcal{J}$ *is executed with speed* s_j*, the total execution time* T^* *of all the jobs is*

$$T^* = \sum_{J_j \in \mathcal{J}} \frac{w_j}{s_j} = \sum_{I_k \in \mathcal{I}} \left(\min\{m, n_k\} \cdot |I_k| \right)$$

The above corollary indicates the total amount of flow that has to be sent from the source node to the destination node in the graph G, concluding the formulation of $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$ as a convex cost flow problem.

Our algorithm for $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$ can be summarized as follows.

Algorithm 3.2

1: Construct the corresponding graph G.
2: Find a convex cost (s,t)-flow $\mathcal{F}$ of value $\sum_{I_k \in \mathcal{I}}(\min\{m, n_k\} \cdot |I_k|)$ in G.
3: Determine the processing time p_j of each job.
4: Apply an algorithm for $\mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|-$ to construct a feasible schedule with respect to p_j's.

In order to establish the optimality of our algorithm, we need the following lemma whose proof can be found in [53]. The lemma concerns $\mathbf{P}|\mathbf{r_j} = \mathbf{0}, \mathbf{d_j} = \mathbf{d}, \mathbf{mgtn}|-$ which is the special case of $\mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|-$ in which all jobs have a common release date and a common deadline.

Lemma 3.6. *An instance of* $P|pmtn, r_j = 0, d_j = d|-$ *is feasible if and only if*

- $p_j \leq d$*, for each* $J_j \in \mathcal{J}$*, and*
- $\sum_{J_j \in \mathcal{J}} p_j \leq m \cdot d$.

Now, we are ready to prove that our algorithm is indeed optimal.

Theorem 3.3. *Algorithm 3.2 produces an optimal schedule for* $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$.

Proof. Initially, we claim that there is a feasible schedule for $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$ such that the total execution time of all the jobs in $\mathcal{J}$ is equal to T if and only if there is a feasible (s,t)-flow of value T in G, for any $T > 0$.

Assume that there exists a feasible schedule for $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$ with total execution time equal to T. Let p_j be the execution time of job J_j in this schedule and let $p_{j,k}$ be the total amount of time that J_j is processed by any processor during the interval I_k. Consider the following (s,t)-flow $\mathcal{F}$ in G.

- $f_{(s,J_j)} = p_j$, for all $J_j \in \mathcal{J}$,
- $f_{(J_j,I_k)} = p_{j,k}$, for all $I_k \in \mathcal{I}$ and $J_j \in \mathcal{A}(I_k)$, and
- $f_{(I_k,t)} = \sum_{J_j \in \mathcal{A}_k} p_{j,k}$, for all $I_k \in \mathcal{I}$.

Since the parallel execution of a job is not permitted, for each job $J_j \in \mathcal{J}$ and each interval $I_k \subseteq [r_j, d_j)$, it holds that $p_{j,k} \leq |I_k|$. Note that each processor $P_i \in \mathcal{P}$ can execute at most one job per unit of time and, as a result, it can operate for at most $|I_k|$ units of time during any interval $I_k \in \mathcal{I}$. Additionally, we can have at most $\min\{m, n_k\}$ processors running at each time. Hence, for each $I_k \in \mathcal{I}$, it holds that $\sum_{J_j \in \mathcal{A}_k} p_{j,k} \leq \min\{m, n_k\} \cdot |I_k|$. We conclude that $\mathcal{F}$ is a feasible (s,t)-flow in G because the capacity of any arc $e \in A$ is not exceeded.

Assume, now, that there is a feasible (s,t)-flow $\mathcal{F}$ of value T in G. Let f_e be the amount of flow that crosses the arc $e \in \mathcal{A}$ according to $\mathcal{F}$. In order to define a feasible schedule $\mathcal{S}$ for $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$, we assign to each job $J_j \in \mathcal{J}$ a speed $s_j = \frac{w_j}{f_{(s,J_j)}}$. So, the total execution time of J_j is $f_{(s,J_j)}$. Moreover, for each interval $I_k \in \mathcal{I}$ and job $J_j \in \mathcal{A}(I_k)$ we set the execution time of J_j during I_k to be $f_{(J_j,I_k)}$. Consider, now, any interval $I_k \in \mathcal{I}$ and let $p_{j,k}$ be the total time that J_j is processed by any processor during I_k in $\mathcal{S}$. Since $\mathcal{F}$ is a feasible (s,t)-flow, it holds that $p_{j,k} = f_{(J_j,I_k)} \leq |I_k|$ and $\sum_{J_j \in \mathcal{A}(I_k)} p_{j,k} = f_{(I_k,t)} \leq \min\{m, n_k\} \cdot |I_k|$. By Lemma 3.6, for each interval $I_k \in \mathcal{I}$, we can schedule the parts of the jobs during I_k feasibly. Thus, we can construct a feasible schedule $\mathcal{S}$ for the whole instance of $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$.

Next, we elaborate on the optimality of our algorithm. By the minimum convex cost flow computation, the algorithm finds an (s,t)-flow of value $T^* = \sum_{I_k \in \mathcal{I}} (\min\{m, n_k\} \cdot |I_k|)$ such that the term $\sum_{J_j \in \mathcal{J}} \frac{w_j^\alpha}{f_{(s,J_j)}^{\alpha-1}}$ is minimized. By our previous claim, we may produce a feasible schedule of total execution time $\sum_{I_k \in \mathcal{I}} (\min\{m, n_k\} \cdot |I_k|)$ such that each job $J_j \in \mathcal{J}$ is assigned a speed $\frac{w_j}{f_{(s,J_j)}}$. Since the energy consumption of this schedule is equal to $\sum_{J_j \in \mathcal{J}} \frac{w_j^\alpha}{f_{(s,J_j)}^{\alpha-1}}$, it is a minimum energy schedule among the schedules of total execution time T^*. By Corollary 3.1, there is always an optimal schedule for $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$ with total execution time T^*. Therefore, the schedule returned by the algorithm is optimal for $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$. □

3.2 Energy Minimization without Migrations or Preemptions

In this section, we consider the non-migratory non-preemptive problem $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}|\mathbf{E}$ of minimizing the energy of a set of jobs that have to be executed by a set of parallel processors and we propose a $(2 - \frac{1}{m})^{\alpha-1}$-approximation algorithm for agreeable instances.

An instance of the problem consists of a set of n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ and a set of m parallel processors $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$. Each job $J_j \in \mathcal{J}$ is specified by an amount of work w_j, a release date r_j and a deadline d_j. The objective is to find a schedule of minimum energy consumption such that each job $J_j \in \mathcal{J}$ is executed during the interval

$[r_j, d_j)$. Note that a set of jobs are agreeable if, for any couple of jobs $J_j, J_{j'} \in \mathcal{J}$ such that $r_j < r_{j'}$, it holds that $d_j \leq d_{j'}$. In this problem, we do not allow preemptions and migrations of jobs, i.e. each job has to be executed without interruptions by a single processor.

Our algorithm creates first an optimal multiprocessor migratory schedule $\mathcal{S}^*_{pr}$ by using an optimal algorithm for $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$ as a black box. Then, it uses the processing time p^*_j of each job J_j in $\mathcal{S}^*_{pr}$ in order to define an appropriate processing time p_j for J_j. In the algorithm's schedule, each job $J_j \in \mathcal{J}$ is executed with a constant speed s_j such that its processing time is equal to p_j. Next, the algorithm schedules the jobs in $\mathcal{J}$ non-preemptively with respect to these processing times according to the Earliest Deadline First (EDF) policy, i.e. at each time that a processor becomes idle, the non-scheduled job with the minimum deadline is scheduled on it for p_j units of time without being interrupted. The choice of the values p_j, $J_j \in \mathcal{J}$, has been made in such a way that the algorithm completes all the jobs before their deadlines. Our algorithm is given in Algorithm 3.3.

Algorithm 3.3

1: Create an optimal migratory schedule $\mathcal{S}^*_{pr}$.
2: Let p^*_j be the total execution time of the job J_j in $\mathcal{S}^*_{pr}$.
3: Set the processing time of each job J_j equal to $p_j = p^*_j/(2 - \frac{1}{m})$.
4: Schedule the jobs in $\mathcal{J}$ non-preemptively according to the Earliest Deadline First (EDF) policy with respect to the p_j's.

Theorem 3.4. *The Algorithm 3.3 produces a* $(2 - \frac{1}{m})^{\alpha-1}$*-approximate solution for the problem* $\mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{agrbl}|\mathbf{E}$.

Proof. Let $\mathcal{S}_{npr}$ be the schedule produced by the Algorithm 3.3. We consider the jobs indexed in non-decreasing order of their release dates/deadlines. That is, for every couple of jobs $J_j, J_{j'} \in \mathcal{J}$ such that $j < j'$, we have that $r_j \leq r_{j'}$ and $d_j \leq d_{j'}$. In what follows, we denote by B_j the beginning time of the job $J_j \in \mathcal{J}$ in $\mathcal{S}_{npr}$. Hence, the completion time C_j of J_j in $\mathcal{S}_{npr}$ is $C_j = B_j + p_j$.

First, we show that $\mathcal{S}_{npr}$ is a feasible schedule. In other words, we will prove that for the completion time of any job $J_j \in \mathcal{J}$, it holds that $C_j \leq d_j$. Once we have established the correctness of our algorithm, then we elaborate on its approximation ratio.

Let us introduce some additional notation. Note that, at each time, either all processors execute some job or there is at least one processor which is idle. Based on this observation, we partition $\mathcal{S}_{npr}$ into maximal intervals: the *full* and the *non-full* intervals. At every time in a *full* interval, every processor executes some job. On the other hand, at each time during a *non-full* interval, there is at least one processor which is idle.

Let τ be the number of the non-full intervals. Let $[u_k, t_k)$, $1 \leq k \leq \tau$, be the k-th non-full interval. Hence, $[t_{k-1}, u_k)$, $1 \leq k \leq \tau + 1$, is a full interval. For convenience, $t_0 = 0$ and $u_{\tau+1} = \max_{J_j \in \mathcal{J}}\{C_j\}$. Note that the schedule $\mathcal{S}_{npr}$ can start with a non-full interval, i.e. $t_0 = u_1$, or it can end with a non-full interval, i.e. $t_\tau = u_{\tau+1}$.

Consider first a job $J_j \in \mathcal{J}$ that is released during a *non-full* interval $[u_k, t_k)$. Since the jobs are scheduled according to EDF policy, J_j starts its execution at its release date

in $\mathcal{S}_{npr}$, i.e. $B_j = r_j$. Given that J_j has smaller processing time in $\mathcal{S}_{npr}$ than in $\mathcal{S}^*_{pr}$ and as $\mathcal{S}^*_{pr}$ is a feasible schedule, it holds that $C_j \leq d_j$.

Consider now a job $J_j \in \mathcal{J}$ which is released during a full interval $[t_k, u_{k+1})$. We denote by $\mathcal{R}_k = \{J_j \in \mathcal{J} : r_j < t_k\}$ the set of jobs which are released before t_k. Let $T_{npr,k}$ be the total amount of time that the jobs in $\mathcal{R}_k$ are executed from t_k and after in $\mathcal{S}_{npr}$ and $T^*_{pr,k}$ be the total amount of time that the jobs in $\mathcal{R}_k$ are executed from t_k and after in $\mathcal{S}^*_{pr}$. In order to go on, we need the following claim whose proof is given after the proof of the theorem for ease of presentation.

Claim 3.1. *For each k, $0 \leq k \leq \tau$, it holds that*

$$T_{npr,k} \leq \frac{T^*_{pr,k}}{(2 - \frac{1}{m})}$$

Let J_f be the first job which is released at t_k or after. Obviously, $r_f = t_k$. For the job J_j, because of our previous claim, we have that

$$C_j \leq t_k + \frac{T_{npr,k} + \sum_{j'=f}^{j-1} p_{j'}}{m} + p_j \leq t_k + \frac{\frac{T^*_{pr,k} + \sum_{j'=f}^{j-1} p^*_{j'}}{m} + p^*_j}{(2 - \frac{1}{m})}$$

As $\mathcal{S}^*_{pr}$ is a feasible schedule, all jobs $J_f, \ldots, J_j$ are executed inside the interval $[t_k, d_j)$ in $\mathcal{S}^*_{pr}$ and $T^*_{pr,k}$ amount of time of the jobs in $\mathcal{R}_k$ is also executed in the same time interval. Therefore, it holds that $T^*_{pr,k} + \sum_{j'=f}^{j} p^*_{j'} \leq m(d_j - t_k)$ and $p^*_j \leq d_j - t_k$. So, we have that

$$\frac{T^*_{pr,k} + \sum_{j'=f}^{j-1} p^*_{j'}}{m} + p^*_j = \frac{T^*_{pr,k} + \sum_{j'=f}^{j} p^*_{j'}}{m} + \left(1 - \frac{1}{m}\right) p^*_j \leq \left(2 - \frac{1}{m}\right)(d_j - t_k)$$

We conclude that

$$C_j \leq t_k + \frac{(2 - \frac{1}{m})(d_j - t_k)}{(2 - \frac{1}{m})} = d_j$$

and, as a result, the schedule $\mathcal{S}_{npr}$ is indeed feasible.

Finally, we elaborate on the approximation ratio of our algorithm. Let $\mathcal{S}^*_{npr}$ be an optimal non-preemptive schedule for our problem. We denote by E_{npr}, E^*_{npr} and E^*_{pr} the total energy consumptions of the schedules $\mathcal{S}_{npr}$, $\mathcal{S}^*_{npr}$ and $\mathcal{S}^*_{pr}$, respectively. When dividing the execution time of all jobs by $(2 - \frac{1}{m})$, at the same time, the speed of each job is multiplied by the same factor. Hence, we have that

$$E_{npr} \leq \left(2 - \frac{1}{m}\right)^{\alpha-1} E^*_{pr} \leq \left(2 - \frac{1}{m}\right)^{\alpha-1} E^*_{npr}$$

Note that the last inequality comes from the fact that the energy consumption E^*_{pr} of an optimal preemptive schedule is always a lower bound on the energy consumption E^*_{npr} of the optimal non-preemptive schedule. The theorem follows. □

Proof of Claim 3.1

Next, we prove the Claim 3.1 that we needed in order to prove the Theorem 3.4.

Proof. We prove the claim by induction to k.

For the induction basis, we have two cases. If $t_0 \neq u_1$, then $T_{npr,0} = T^*_{pr,0} = 0$. If $t_0 = u_1$, then the schedule begins with a non-full interval. In this case, we have to consider the jobs in $\mathcal{R}_1$ for the induction basis. Since the jobs are scheduled according to the EDF policy in $\mathcal{S}_{npr}$, every job $J_j \in \mathcal{R}_1$ starts at its release date, i.e. $B_j = r_j$. Given that $p_j = p^*_j/(2 - \frac{1}{m})$ and that $\mathcal{S}^*_{pr}$ is a feasible schedule, the claim holds.

For our induction step, assume that the claim is true for $1, 2, \ldots, k$. We will show that $T_{npr,k+1} \leq T^*_{pr,k+1}/(2 - \frac{1}{m})$. We consider two cases.

Case 1: $u_{k+1} \geq \frac{(1-\frac{1}{m})t_{k+1}+t_k}{(2-\frac{1}{m})}$.

Recall that $\mathcal{R}_{k+1}$ is the set of jobs with $r_j < t_{k+1}$ in $\mathcal{S}_{npr}$. We partition the jobs in $\mathcal{R}_{k+1}$ such that $C_j > t_{k+1}$ into the following three subsets:

- $\mathcal{A}$: the jobs with $B_j < t_k$,
- $\mathcal{B}$: the jobs with $t_k \leq B_j < u_{k+1}$, and
- $\mathcal{C}$: the jobs with $u_{k+1} \leq B_j < t_{k+1}$.

Let $T(\mathcal{A})$ be the total amount of time that the jobs in $\mathcal{A}$ are executed after t_k. Obviously, $T(\mathcal{A}) \leq T_{npr,k}$ as $\mathcal{A} \subseteq \mathcal{R}_k$. Since the schedule $\mathcal{S}_{npr}$ is non-preemptive, each job $J_j \in \mathcal{A}$ is processed for $t_{k+1} - t_k$ units of time during $[t_k, t_{k+1})$. Thus, we have that

$$T_{npr,k+1} = (T(\mathcal{A}) - |\mathcal{A}|(t_{k+1} - t_k)) + \sum_{J_j \in \mathcal{B}} (B_j + p_j - t_{k+1}) + \sum_{J_j \in \mathcal{C}} (B_j + p_j - t_{k+1})$$

In the extreme case, all the processors execute some job of $\mathcal{R}_{k+1}$ during the interval $[t_k, t_{k+1})$ in the schedule $\mathcal{S}^*_{pr}$. So, we have that

$$\begin{aligned} T^*_{pr,k+1} &\geq T^*_{pr,k} + \sum_{J_j \in \mathcal{R}_{k+1} \setminus \mathcal{R}_k} p^*_j - m \cdot (t_{k+1} - t_k) \\ &\geq \left(2 - \frac{1}{m}\right) \left(T_{npr,k} + \sum_{J_j \in \mathcal{R}_{k+1} \setminus \mathcal{R}_k} p_j \right) - m \cdot (t_{k+1} - t_k) \end{aligned}$$

The second inequality comes from our induction hypothesis and the way we obtained the processing times in $\mathcal{S}_{npr}$ from $\mathcal{S}^*_{pr}$. Note that the amount of time $(T_{npr,k} + \sum_{J_j \in \mathcal{R}_{k+1} \setminus \mathcal{R}_k} p_j)$ is the total amount of time during which the jobs in $\mathcal{R}_{k+1}$ are executed from t_k and after in $\mathcal{S}_{npr}$. By definition, this amount is $T(\mathcal{A})$ for the jobs in $\mathcal{A}$. Recall that these jobs have $B_j < t_k$ and $C_j > t_{k+1}$ and hence $|\mathcal{A}|$ processors are dedicated to them during the interval $[t_k, t_{k+1})$. Consider the set of jobs not in $\mathcal{A}$, which are released before u_{k+1} and are completed after t_k. These jobs contribute to $(T_{npr,k} + \sum_{J_j \in \mathcal{R}_{k+1} \setminus \mathcal{R}_k} p_j)$ with at least $(m - |\mathcal{A}| - |\mathcal{B}|)(u_{k+1} - t_k) + \sum_{J_j \in \mathcal{B}} (B_j + p_j - t_k)$ amount of time, since there is no idle

period in the interval $[t_k, u_{k+1})$. Finally, for the jobs in $\mathcal{C}$ this contribution is $\sum_{J_j \in \mathcal{C}} p_j$. Hence,

$$\begin{aligned} T^*_{pr,k+1} \geq & \left(2 - \frac{1}{m}\right)\left(T(\mathcal{A}) + (m - |\mathcal{A}| - |\mathcal{B}|)(u_{k+1} - t_k) + \sum_{J_j \in \mathcal{B}} (B_j + p_j - t_k) + \sum_{J_j \in \mathcal{C}} p_j\right) \\ & -m \cdot (t_{k+1} - t_k) \end{aligned}$$

Thus, we have

$$\begin{aligned} \frac{T^*_{pr,k+1}}{(2-\frac{1}{m})} - T_{npr,k+1} \geq & (m - |\mathcal{A}| - |\mathcal{B}|)(u_{k+1} - t_k) - \sum_{J_j \in \mathcal{B}} t_k - \frac{m(t_{k+1} - t_k)}{2 - \frac{1}{m}} \\ & +|\mathcal{A}|(t_{k+1} - t_k) + \sum_{J_j \in \mathcal{B}} t_{k+1} - \sum_{J_j \in \mathcal{C}} (B_j - t_{k+1}) \\ = & u_{k+1}(m - |\mathcal{A}| - |\mathcal{B}|) - m\left(t_k + \frac{t_{k+1} - t_k}{2 - \frac{1}{m}}\right) \\ & +(|\mathcal{A}| + |\mathcal{B}|)t_{k+1} + \sum_{J_j \in \mathcal{C}} (t_{k+1} - B_j) \\ \geq & \left(\frac{(1-\frac{1}{m})t_{k+1} + t_k}{2 - \frac{1}{m}}\right)(m - |\mathcal{A}| - |\mathcal{B}|) - m\left(t_k + \frac{t_{k+1} - t_k}{2 - \frac{1}{m}}\right) \\ & +(|\mathcal{A}| + |\mathcal{B}|)t_{k+1} \end{aligned}$$

where the last inequality follows from the fact that $t_{k+1} \geq B_j$ for each job in $\mathcal{C}$ and using our assumption for the case we consider. Note that $|\mathcal{A}| + |\mathcal{B}| \geq 1$ as otherwise $T_{npr,k+1}$ consists only of jobs in $\mathcal{C}$ which are scheduled at their release date in $\mathcal{S}_{npr}$ and the claim holds directly. Therefore,

$$\begin{aligned} \frac{T^*_{pr,k+1}}{(2-\frac{1}{m})} - T_{npr,k+1} \geq & m\left(\frac{(1-\frac{1}{m})t_{k+1} + t_k}{2 - \frac{1}{m}} - t_k - \frac{t_{k+1} - t_k}{2 - \frac{1}{m}}\right) \\ & +(|\mathcal{A}| + |\mathcal{B}|)\left(t_{k+1} - \frac{(1-\frac{1}{m})t_{k+1} + t_k}{2 - \frac{1}{m}}\right) \\ \geq & \frac{t_k - t_{k+1}}{2 - \frac{1}{m}} + \frac{t_{k+1} - t_k}{2 - \frac{1}{m}} \\ \geq & 0 \end{aligned}$$

Case 2: $u_{k+1} < \frac{(1-\frac{1}{m})t_{k+1} + t_k}{(2-\frac{1}{m})}$.

In $\mathcal{S}_{npr}$, for a given job J_j which completes after t_{k+1}, let $p_{j,k+1}$ be the processing time of J_j after the time t_{k+1}. Similarly, let $p^*_{j,k+1}$, be the execution time of J_j after time t_{k+1} in $\mathcal{S}^*_{pr}$. In this case, we partition the jobs of the set $\mathcal{R}_{k+1}$ such that $C_j > t_{k+1}$ as follows:

- $\mathcal{A}$: the jobs with $r_j < t_k$,
- $\mathcal{B}$: the jobs with $t_k \leq r_j < u_{k+1}$, and
- $\mathcal{C}$: the jobs with $u_{k+1} \leq r_j < t_{k+1}$.

Consider the jobs in $\mathcal{A}$. Recall that the jobs are scheduled by the algorithm according to EDF. Since the instance is agreeable, this means that the earliest released jobs are scheduled first. So, the jobs in $\mathcal{A}$ start before the jobs in $\mathcal{B}$ and $\mathcal{C}$ in the algorithms schedule. This, combined with the induction hypothesis yields that $\sum_{J_j \in \mathcal{A}} p_{j,k+1} \leq \sum_{J_j \in \mathcal{A}} p^*_{j,k+1}/(2 - \frac{1}{m})$.

Consider a job $J_j \in \mathcal{B}$. Clearly, for such a job it holds that $B_j < u_{k+1}$ and $C_j > t_{k+1}$ in $\mathcal{S}_{npr}$. We will show that $p^*_j > t_{k+1} - t_k$. Assume for contradiction that $p^*_j \leq t_{k+1} - t_k$. Hence, we have that

$$\begin{aligned} p_{j,k+1} &= B_j + p_j - t_{k+1} \\ &= B_j + \frac{p^*_j}{(2 - \frac{1}{m})} - t_{k+1} \\ &< u_{k+1} + \frac{p^*_j}{(2 - \frac{1}{m})} - t_{k+1} \\ &< \frac{(1 - \frac{1}{m})t_{k+1} + t_k}{(2 - \frac{1}{m})} + \frac{t_{k+1} - t_k}{(2 - \frac{1}{m})} - t_{k+1} \\ &= 0 \end{aligned}$$

which is a contradiction as, by definition, it must be the case that $p_{j,k+1} > 0$. Thus, for the job J_j, we have that $p^*_{j,k+1} \geq p^*_j + t_k - t_{k+1}$ So,

$$\begin{aligned} \frac{p^*_{j,k+1}}{(2 - \frac{1}{m})} - p_{j,k+1} &= \frac{p^*_j + t_k - t_{k+1}}{(2 - \frac{1}{m})} - (B_j + p_j - t_{k+1}) \\ &= p_j + \frac{t_k - t_{k+1}}{(2 - \frac{1}{m})} - B_j - p_j + t_{k+1} \\ &\geq \frac{t_k - t_{k+1}}{(2 - \frac{1}{m})} - u_{k+1} + t_{k+1} \\ &= \frac{(1 - \frac{1}{m})t_{k+1} + t_k}{(2 - \frac{1}{m})} - u_{k+1} > 0 \end{aligned}$$

as $B_j < u_{k+1}$ and $u_{k+1} < \frac{(1-\frac{1}{m})t_{k+1}+t_k}{(2-\frac{1}{m})}$.

Consider now a job $J_j \in \mathcal{C}$. This job starts its execution at its release date in $\mathcal{S}_{npr}$, i.e. $B_j = r_j$. Given that J_j has smaller processing time in $\mathcal{S}_{npr}$ than in $\mathcal{S}^*_{pr}$ and that $\mathcal{S}^*_{pr}$ is feasible, we have that $p^*_{j,k} - p_{j,k} > 0$.

Summing up for all jobs in $\mathcal{A} \cup \mathcal{B} \cup \mathcal{C}$, we get $T_{npr,k} \leq \frac{T^*_{pr,k}}{(2-\frac{1}{m})}$, and the claim follows. □

Chapter 4

Heterogeneous Environments

In this chapter, we study multiprocessor scheduling problems on heterogeneous environments. In such environments, we have a set of processors which run in parallel and they obey to different speed-to-power functions. Moreover, the jobs have processor dependent works, release dates and deadlines.

Initially, in Section 4.1, we propose a near optimal polynomial time algorithm for the energy minimization problem $\mathbf{S}, \mathbf{R}|\mathbf{w_{i,j}}, \mathbf{r_{i,j}}, \mathbf{d_{i,j}}, \mathbf{mgtn}|\mathbf{E}$, where preemptions and migrations of jobs are allowed. This algorithm is based on solving a configuration Linear Program (LP) with the Ellipsoid algorithm.

Next, in Section 4.2, we consider the problem $\mathbf{S}, \mathbf{R}|\mathbf{w_{i,j}}, \mathbf{r_{i,j}}, \mathbf{d_{i,j}}, \mathbf{pmtn}|\mathbf{E}$ of minimizing the energy, where preemptions of jobs are allowed but migrations are forbidden. We formulate this problem as an integer configuration LP and we show how to obtain a constant factor approximate solution for this LP by solving its fractional relaxation and applying randomized rounding. In order to improve the running time of our algorithm we also formulate the problem as a compact integer LP and we show that we can obtain a solution for the fractional relaxation of the configuration LP by solving the fractional relaxation of the compact LP.

Finally, in Section 4.3, we address the problem of minimizing the average completion time plus energy, i.e. $\mathbf{S}, \mathbf{R}|\mathbf{w_{i,j}}| \sum \mathbf{C_j} + \beta \mathbf{E}$, and we propose an optimal polynomial time algorithm which is based on the formulation of the problem as a minimum weighted perfect matching problem.

4.1 Energy Minimization with Migrations and Preemptions

In this section, we consider the problem $\mathbf{S}, \mathbf{R}|\mathbf{w_{i,j}}, \mathbf{r_{i,j}}, \mathbf{d_{i,j}}, \mathbf{mgtn}|\mathbf{E}$ and we propose a near-optimal algorithm which returns a schedule with energy consumption at most $OPT + \epsilon$, where OPT is the energy consumption of an optimal solution. The algorithm is polynomial to the size of the instance and $1/\epsilon$.

In the problem $\mathbf{S}, \mathbf{R}|\mathbf{w_{i,j}}, \mathbf{r_{i,j}}, \mathbf{d_{i,j}}, \mathbf{mgtn}|\mathbf{E}$, we have a set of n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ which have to be executed by a set of m parallel processors $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$. Each job $J_j \in \mathcal{J}$ has a processor-dependent work $w_{i,j}$, release date $r_{i,j}$ and deadline $d_{i,j}$ on

every processor $P_i \in \mathcal{P}$. The work $w_{i,j}$ is the amount of work that must be executed for J_j if it is executed entirely by the processor P_i. Note that a job $J_j \in \mathcal{J}$ can be executed on $P_i \in \mathcal{P}$ only during the time interval $[r_{i,j}, d_{i,j})$ and we say that J_j is *active* on the processor P_i during this interval. The processor $P_i \in \mathcal{P}$ satisfies the speed-to-power function $Q_i = s^{\alpha_i}$, where $\alpha_i > 1$. Assume that the amount of work executed for the job J_j on the processor P_i is equal to w. Then, the portion of J_j executed on P_i is equal to $\frac{w}{w_{i,j}}$. A job is completed only when the total portion executed for it on all the processors is equal to 1. The objective is to find a feasible schedule of minimum energy consumption.

We first formulate the problem as a configuration Linear Program (LP) with an exponential number of variables and a polynomial number of constraints. Such an LP cannot be solved directly in polynomial time by applying to it an existing algorithm for linear programming. However, we can obtain a polynomial-time algorithm by solving its dual LP with the Ellipsoid algorithm as we describe in the remainder of this section.

Let us, first, formulate the problem as a configuration LP. In order to do so, we have to define the notion of a configuration for this problem. We define a configuration c as an one-to-one assignment of x jobs, $0 \leq x \leq m$, to the m processors as well as an assignment of a speed value to every processor. Note that some processors may be idle according to c and their speed is zero. A well defined schedule for our problem has to specify exactly one configuration at each time t. An example of a configuration is illustrated in Figure 4.1.

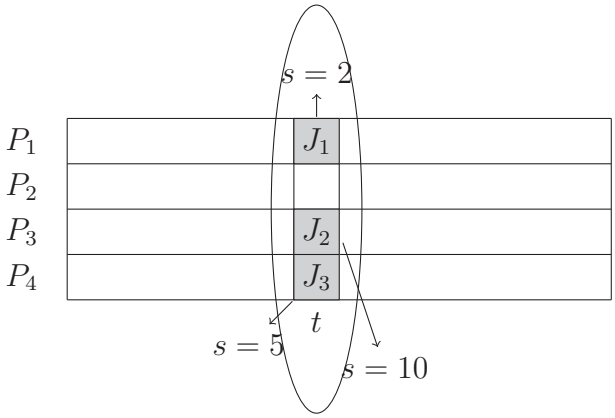


Figure 4.1: An example of a configuration for an instance with four processors. Note that the processor P_2 executes no job and its speed is equal to zero according to this configuration.

We denote by $\mathcal{C}$ the set of all possible configurations. Clearly, the cardinality of $\mathcal{C}$ is unbounded, since the speed of a processor can be any non-negative real value. So, we discretize the set of possible speed values and we consider only a finite number of speeds at which the processors can run, based on Lemma 4.1.

Lemma 4.1. *There is a feasible schedule of energy consumption at most $OPT + \epsilon$ that uses a finite (exponential to the size of the instance and $1/\epsilon$) number of discrete processors' speeds.*

Proof. To discretize the speeds, we first define a lower and an upper bound on the speed of any processor in any optimal schedule. For the lower bound, consider a job $J_j \in \mathcal{J}$. Recall that the release dates and the deadlines of J_j are different on different processors. Hence, the feasible intervals of J_j on different processors may be completely disjoint, that is the processing time of J_j in an optimal schedule can be at most $\sum_{P_i \in \mathcal{P}}(d_{i,j} - r_{i,j})$. Therefore, due to the convexity of the speed-to-power function, a lower bound on the speed of every processor is

$$s_{LB} = \min_{J_j \in \mathcal{J}} \left\{ \frac{\min_{P_i \in \mathcal{P}}\{w_{i,j}\}}{\sum_{P_i \in \mathcal{P}}(d_{i,j} - r_{i,j})} \right\}$$

For the upper bound, we consider the case where all the jobs are executed in the minimum active interval of any job, i.e. $\min_{J_j \in \mathcal{J}}\{d_{i,j} - r_{i,j}\}$. Hence, an upper bound on the speed of every processor is

$$s_{UB} = \frac{\sum_{J_j \in \mathcal{J}} \max_{P_i \in \mathcal{P}}\{w_{i,j}\}}{\min_{J_j \in \mathcal{J}}\{d_{i,j} - r_{i,j}\}}$$

Given these lower and upper bounds and a small constant $\eta > 0$, we discretize the speed values in a geometric way. In other words, we consider only the speeds of the form $s_{LB}, (1+\eta)s_{LB}, (1+\eta)^2 s_{LB}, \ldots, (1+\eta)^k s_{LB}$, where k is the smallest integer such that $(1+\eta)^k s_{LB} \geq s_{UB}$. Hence, the number of speed values is equal to $k = \log_{1+\eta} \frac{s_{UB}}{s_{LB}}$, which is polynomial to the size of the instance and to $1/\log(1+\eta)$.

Consider now an optimal schedule for our problem. Let $\mathcal{S}$ be the schedule obtained from the optimal one by rounding up the processors' speeds to the closest discrete value. The ratio of the energy consumption of any processor $P_i \in \mathcal{P}$ at any time t in $\mathcal{S}$ over the energy consumption of P_i at t in the optimal schedule is at most $(1+\eta)^{\alpha_i}$. By summing up for all processors and all times, we get that the energy consumption of $\mathcal{S}$ is at most $(1+\eta)^{\alpha_{max}} OPT$. Finally, if we pick a value η such that $\eta = (1 + \frac{\epsilon}{OPT})^{1/\alpha_{max}} - 1$, then the energy consumption of $\mathcal{S}$ is at most $OPT + \epsilon$. With this selection of η, the number of discrete speeds is, in the worst case, exponential to the size of the instance and $1/\epsilon$. □

In what follows, we only consder schedules that satisfy Lemma 4.1. Let $t_0 < t_1 < \ldots < t_\tau$ be the time instants that correspond to release dates and deadlines of jobs so that there is a time t_k for every possible release date and deadline. We denote by $\mathcal{I}$ the set of all possible intervals of the form $[t_{k-1}, t_k)$, for $1 \leq k \leq \tau$. Let $|I|$ be the length of the interval I.

In order to formulate our problem as a configuration LP, we introduce a variable $x_{I,c}$, for each $I \in \mathcal{I}$ and $c \in \mathcal{C}$, which corresponds to the total processing time during the interval $I \in \mathcal{I}$ that the processors run according to the configuration $c \in \mathcal{C}$. We denote by E_c the instantaneous energy consumption of the processors if they run with respect to the configuration c. Moreover, let $s_{j,c}$ be the speed of the job J_j according to the configuration c. We denote by $\mathcal{A}(I, c)$ the set of jobs which are active during the interval I and which are executed on some processor by the configuration c. Finally, let $P_{i(j,c)}$ be the processor in which the job J_j is assigned by the configuration c. Then, we propose

the following configuration LP.

$$\begin{aligned}
\min \sum_{I\in\mathcal{I}, c\in\mathcal{C}} & E_c \cdot x_{I,c} & & \\
\sum_{c\in\mathcal{C}} & x_{I,c} \leq |I| & I \in \mathcal{I} & \quad (4.1)\\
\sum_{I,c:\ J_j\in\mathcal{A}(I,c)} & \frac{s_{j,c}}{w_{i(j,c),j}} x_{I,c} \geq 1 & J_j \in \mathcal{J} & \quad (4.2)\\
& x_{I,c} \geq 0 & I \in \mathcal{I},\ c \in \mathcal{C} &
\end{aligned}$$

Consider the schedule for the interval I that occurs by an arbitrary order of the configurations assigned to I. This schedule is feasible, as the processing time of all configurations assigned to I is equal to the length of the interval. Hence, Inequality (4.1) ensures that for each interval I there is exactly one configuration for each time $t \in I$. Inequality (4.2) implies that each job J_j is entirely executed.

The above configuration LP has an exponential number of variables and a polynomial number of constraints. We associate to the constraints (4.1) and (4.2) the dual variables μ_I and λ_j, respectively. So, we obtain its dual LP which follows.

$$\begin{aligned}
\max \sum_{J_j\in\mathcal{J}} & \lambda_j - \sum_{I\in\mathcal{I}} \mu_I |I| & & \\
\sum_{J_j\in\mathcal{A}(I,c)} & \frac{s_{j,c}}{w_{i(j,c),j}} \lambda_j - \mu_I \leq E_c & I \in \mathcal{I},\ c \in \mathcal{C} & \\
& \mu_I, \lambda_j \geq 0 & I \in \mathcal{I},\ J_j \in \mathcal{J} &
\end{aligned}$$

The above dual LP has polynomial number of variables and an exponential number of constraints. A well-known fact in Combinatorial Optimization is that we may solve such LPs in polynomial time by applying the Ellipsoid algorithm. However, we need a *polynomial-time separation oracle*, i.e. a polynomial algorithm which, given any solution for the LP, it decides if this solution is feasible and, if not, it identifies a violated constraint (which is not satisfied by the solution). Next, we show that the dual LP is polynomially solvable because it admits a polynomial-time separation oracle.

The separation oracle for the dual LP works as follows. For each $I \in \mathcal{I}$, we try to find if there is a violated constraint. For a given I, it suffices to check the minimum among the values $E_c - \sum_{J_j\in\mathcal{A}(I,c)} \frac{s_{j,c}}{w_{i(j,c),j}} \lambda_j$ among all possible configurations c. If this minimum value is less than $-\mu_I$, then we have a violated constraint. Otherwise, if we cannot find any violated constraint for all $I \in \mathcal{I}$, then the dual solution is feasible.

Recall that $E_c = \sum_{J_j\in\mathcal{A}(I,c)} s_{j,c}^{\alpha_{i(j,c)}}$, and hence we want to find the minimum value of $\sum_{J_j\in\mathcal{A}(I,c)} (s_{j,c}^{\alpha_{i(j,c)}} - \frac{s_{j,c}}{w_{i(j,c),j}} \lambda_j)$. For each job $J_j \in \mathcal{J}$ that is active during I, the term $s_{j,c}^{\alpha_{i(j,c)}} - \frac{s_{j,c}}{w_{i(j,c),j}} \lambda_j$ is minimized at the discrete value $v_{i(j,c),j}$ which is one of the two closest possible discrete speeds to the value $\left(\frac{\lambda_j}{\alpha_{i(j,c)} \cdot w_{i(j,c),j}}\right)^{1/(\alpha_{i(j,c)}-1)}$. To see this we just need to notice that we minimize an one variable convex function over a set of possible discrete values. The value $\left(\frac{\lambda_j}{\alpha_{i(j,c)} \cdot w_{i(j,c),j}}\right)^{1/(\alpha_{i(j,c)}-1)}$ is obtained by minimizing $s_{j,c}^{\alpha_{i(j,c)}} - \frac{s_{j,c}}{w_{i(j,c),j}} \lambda_j$ if

there is no discretization of the speeds and it is obtained by equating the derivative of the last expression with zero. Hence, for each interval $I \in \mathcal{I}$, we want to find a configuration c that minimizes $\sum_{J_j \in \mathcal{A}(I,c)} (v_{i(j,c),j}^{\alpha_{i(j,c)}} - \frac{v_{i(j,c),j}}{w_{i(j,c),j}} \lambda_j)$.

Since a configuration c assigns $0 \leq x \leq m$ jobs to m processors, the problem of minimizing the last expression reduces to a minimum weighted matching problem on the bipartite graph which is constructed as follows. We introduce one node for each job and one node for each processor. There is an edge between each job $J_j \in \mathcal{J}$, which is active during the interval I, and each processor $P_i \in \mathcal{P}$ with weight equal to $(v_{i,j}^{\alpha_i} - \frac{v_{i,j}}{w_{i,j}} \lambda_j)$. A minimum weighted matching in such a bipartite graph defines a configuration c, that is an assignment of $x \leq m$ jobs to m processors with their speed values.

Hence, there is a polynomial time separation oracle for the dual problem. To apply the Ellipsoid algorithm in polynomial time, we need to check two additional technical conditions. The first condition is that the values of all dual variables are upper bounded by some number R. The second condition is that there is a feasible point (or solution) for the dual program such that every point in a radius r is feasible. In this case, the running time of the Ellipsoid algorithm is polynomial in $\log \frac{R}{r}$ (see [40]).

The first condition and the bound on R can be derived from the fact that the solution of the problem must be a vertex of the corresponding polyhedron and we know that the value of optimal solution is bounded. The second condition is satisfied for the point (λ, μ) defined as follows. We set $\lambda_j = 1$ for all $J_j \in \mathcal{J}$ and μ_I to be large enough so that $-\mu_I + 1 \leq \min_c \left(E_c - 2 \sum_{j \in (I,c)} \frac{s_{j,c}}{w_{i(j,c),j}} \right)$. Hence, the inequalities are satisfied in the ball of radius 1 around (λ, μ), that is $r = 1$.

As we can compute an optimal solution for the dual LP, we can also find an optimal solution for the primal LP by solving it only with the variables corresponding to the constraints of the dual LP that were found to be violated during the run of the Ellipsoid algorithm to the dual LP and setting all other primal variables to be zero. The number of these violated constraints is polynomial to the size of the instance and $1/\epsilon$. Thus, we can solve the primal LP with a polynomial number of variables and the next theorem follows.

Theorem 4.1. *A schedule for the heterogeneous multiprocessor speed-scaling problem with migrations of energy consumption $OPT + \epsilon$ can be found in polynomial time with respect to the size of the instance and $1/\epsilon$.*

4.2 Energy Minimization without Migrations with Preemptions

In this section we consider the problem $\mathbf{S}, \mathbf{R} | \mathbf{w_{i,j}}, \mathbf{r_{i,j}}, \mathbf{d_{i,j}}, \mathbf{pmtn} | \mathbf{E}$ of scheduling a set of jobs on parallel heterogeneous processors where preemptions of jobs are allowed but migrations are forbidden and we propose a constant factor approximation algorithm which is based on randomized rounding.

In this problem, we have a set of n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ and a set of m parallel heterogeneous processors $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$. The speed-to-power function of the processor P_i is defined as $Q_i = s^{\alpha_i}$. We associate to every job $J_j \in \mathcal{J}$ a work $w_{i,j}$,

a release date $r_{i,j}$ and a deadline $d_{i,j}$, for every $P_i \in \mathcal{P}$. Since migration of jobs is forbidden, every job $J_j \in \mathcal{J}$ must be executed on a single processor among the ones in $\mathcal{P}$. If the job J_j is assigned on the processor P_i, then $w_{i,j}$ units of work must be executed for it during the interval $[r_{i,j}, d_{i,j})$. As we allow preemptions of jobs, a job may be executed, suspended and resumed later from the point of suspension. The objective is to construct a schedule of minimum energy consumption.

We propose a constant factor approximation algorithm for $\mathbf{S}, \mathbf{R}|\mathbf{w_{i,j}}, \mathbf{r_{i,j}}, \mathbf{d_{i,j}}, \mathbf{pmtn}|\mathbf{E}$. Initially, we formulate the problem as an integer configuration Linear Program (LP) with an exponential number of variables and a polynomial number of constraints. Then, we consider the fractional relaxation of this integer LP in which the variables are allowed to take fractional values and we show a way of solving it optimally in polynomial time by applying the Ellipsoid algorithm. Given an optimal solution for the fractional relaxation of the integer LP, we apply randomized rounding to get a feasible integral solution which corresponds to feasible schedule for our problem. At the end of this section, we show that we can obtain a solution for the fractional relaxation of the integer configuration LP by solving a more compact LP. This allows us to use a faster linear programming algorithm instead of the Ellipsoid algorithm.

Integer Configuration LP

In order to formulate our problem as an integer configuration LP, we need to discretize the time. In the following lemma we assume that all the release dates and the deadlines are integers. Let OPT be the energy consumption of an optimal schedule for our problem.

Lemma 4.2. *There is a feasible schedule with energy consumption at most* $((1+\frac{\epsilon}{1-\epsilon})(1+\frac{1}{n-2}))^{\alpha_{max}} \cdot OPT$ *such that, for every job* $J_j \in \mathcal{J}$*, if* J_j *is executed on the processor* P_i*, then each piece of* J_j *starts and ends at a time point* $r_{i,j} + k\frac{\epsilon}{n^3}(d_{i,j} - r_{i,j})$*, where* $k \geq 0$ *is an integer.*

Proof. Consider an optimal schedule $\mathcal{S}^*$ of our problem. We will first transform $\mathcal{S}^*$ to a feasible schedule $\mathcal{S}$ in which the execution time of each job $J_j \in \mathcal{J}$ executed on processor $P_i \in \mathcal{P}$ is at least $\frac{\epsilon}{n}(d_{i,j} - r_{i,j})$ and the number of preemptions is at most n.

As the release dates and the deadlines of the jobs are integers, we can divide the time horizon into unit length slots. Now, we can get the schedule $\mathcal{S}$ from $\mathcal{S}^*$ as follows. We increase the processors' speeds so as to create an idle period of length ϵ inside every unit slot. This can be done by increasing the speeds of all the jobs by a factor of $1 + \frac{\epsilon}{1-\epsilon}$. In this way, the total energy consumption in $\mathcal{S}$ is increased by a factor of $(1 + \frac{\epsilon}{1-\epsilon})^{\alpha_{max}}$. For each job $J_j \in \mathcal{J}$, we reserve a period of length $\frac{\epsilon}{n}$ inside each unit slot of $[r_{i,j}, d_{i,j})$ on the processor by which J_j is executed in $\mathcal{S}^*$. Then, in order to obtain $\mathcal{S}$, we decrease the speed of J_j so that its total work is executed during the periods where J_j was executed in $\mathcal{S}^*$ and the additional $(d_{i,j} - r_{i,j})$ reserved periods. Therefore, in the final schedule the processing time of each job $J_j \in \mathcal{J}$ is at least $\frac{\epsilon}{n}(d_{i,j} - r_{i,j})$. After this transformation we apply the Earliest Deadline First (EDF) policy to each processor separately with respect to the set of jobs assigned on this processor in $\mathcal{S}^*$ and the speeds defined above. This ensures that we have a schedule with at most n preemptions, as in EDF a job may be interrupted only when another job is released.

Next, we transform $\mathcal{S}$ to a new schedule $\mathcal{S}'$ satisfying the statement of the lemma. For each job $J_j \in \mathcal{J}$ which is executed on the processor $P_i \in \mathcal{P}$, we split the interval $[r_{i,j}, d_{i,j})$ into slots of length $\frac{\epsilon}{n^3}(d_{i,j} - r_{i,j})$, i.e. we partition $[r_{i,j}, d_{i,j})$ into intervals of the form $[r_{i,j} + k\frac{\epsilon}{n^3}(d_{i,j} - r_{i,j}), r_{i,j} + (k+1)\frac{\epsilon}{n^3}(d_{i,j} - r_{i,j}))$, where $k \geq 0$ is an integer. As the processing time of J_j in $\mathcal{S}$ is at least $\frac{\epsilon}{n}(d_{i,j} - r_{i,j})$, the execution of J_j has been partitioned into at least n^2 slots. In each of these slots, the job J_j either is executed during the whole slot or is executed into a fraction of it. As we have applied the EDF policy, each job is preempted at most n times, and hence at most $2n$ of these slots are not fully occupied by J_j, since for each preempted piece of J_j at most two slots may not be completely covered by it. We can modify the schedule $\mathcal{S}$ and get the schedule $\mathcal{S}'$ by executing the job J_j is executed only in the slots where it was entirely executed in $\mathcal{S}$. The number of these slots is at least $n^2 - 2n$. Thus, we have to increase the speed of J_j by a factor of at most $1 + \frac{1}{n-2}$, and hence the energy is increased by a factor of $(1 + \frac{1}{n-2})^{\alpha_{max}}$. By taking into account that the energy of $\mathcal{S}$ is a factor of $(1 + \frac{\epsilon}{1-\epsilon})^{\alpha_{max}}$ far from OPT, the lemma follows. □

Let $\mathcal{S}$ be a schedule that satisfies Lemma 4.2 and $J_j \in \mathcal{J}$ be a job executed on the processor $P_i \in \mathcal{P}$ in $\mathcal{S}$. The above lemma implies that the interval $[r_{i,j}, d_{i,j})$ can be partitioned into polynomial, with respect to the size of the instance and $1/\epsilon$, number of equal length slots. In each of these slots, either J_j is executed during the whole slot or is not executed at all. In what follows we consider schedules that satisfy Lemma 4.2.

Let us, now, formulate our problem as an integer configuration LP. A configuration c is a schedule for a single job on a single processor. Specifically, for a given job J_j, a configuration determines the slots, with respect to Lemma 4.2, during which J_j is executed. Given a configuration c for a job $J_j \in \mathcal{J}$, we can compute the processing time of J_j with respect to c which is equal to the number of slots in c multiplied by the length of a slot. Due to the convexity of the speed-to-power function, in a minimum energy schedule that satisfies Lemma 4.2, the job J_j runs with a constant speed s_j. Hence, s_j is equal to the work of J_j over its execution time. Let $\mathcal{C}$ be the set of all possible feasible configurations for all jobs on all processors.

In order to ensure the feasibility of any schedule corresponding to a solution of the integer configuration LP, we need to further partition the time. Given a processor $P_i \in \mathcal{P}$, consider the time points of all jobs of the form $r_{i,j} + k\frac{\epsilon}{n^3}(d_{i,j} - r_{i,j})$ as introduced in Lemma 4.2. Let $t_{i,1}, t_{i,2}, \ldots, t_{i,\ell_i}$ be the ordered sequence of these time points. Consider now the intervals $[t_{i,p}, t_{i,p+1})$, $1 \leq p \leq \ell_i - 1$. In a schedule that satisfies Lemma 4.2, in each such interval either there is exactly one job that is executed during the whole interval or the interval is idle on the processor P_i. Note also that these intervals might not have the same length. Let $\mathcal{I}$ be the set of all these intervals for all processors.

We introduce the binary variable $x_{i,j,c}$ which is equal to one if the job $J_j \in \mathcal{J}$ is entirely executed on the processor $P_i \in \mathcal{P}$ according to the configuration c, and zero otherwise. Note that, given the configuration c and the processor P_i where the job J_j is executed, we can compute the energy consumption $E_{i,j,c}$ for the execution of J_j. For ease of notation, we say that $I \in (i, j, c)$ if the interval $I \in \mathcal{I}$ is included in the configuration $c \in \mathcal{C}$ for the job $J_j \in \mathcal{J}$ on the processor $P_i \in \mathcal{P}$, that is there is a slot $(r_{i,j} + k\frac{\epsilon}{n^3}(d_{i,j} - r_{i,j}), r_{i,j} + (k+1)\frac{\epsilon}{n^3}(d_{i,j} - r_{i,j})]$ in c that contains I. Then, our problem can be formulated as the following integer configuration LP.

$$\min \sum_{P_i \in \mathcal{P}} \sum_{J_j \in \mathcal{J}} \sum_{c \in \mathcal{C}} E_{i,j,c} \cdot x_{i,j,c}$$

$$\sum_{P_i \in \mathcal{P}} \sum_{c \in \mathcal{C}} x_{i,j,c} \geq 1 \qquad J_j \in \mathcal{J} \tag{4.3}$$

$$\sum_{I \in (i,j,c)} x_{i,j,c} \leq 1 \qquad I \in \mathcal{I} \tag{4.4}$$

$$x_{i,j,c} \in \{0,1\} \qquad P_i \in \mathcal{P}, J_j \in \mathcal{J}, c \in \mathcal{C} \tag{4.5}$$

Inequality (4.3) enforces that each job is entirely executed by some configuration. Inequality (4.4) ensures that at most one job is executed in each interval $[t_{i,p}, t_{i,p+1})$, $1 \leq p \leq \ell_i - 1$.

We next relax the constraints (4.5) so that $x_{i,j,c} \geq 0$. As the number of variables of the relaxed LP is exponential to the size of the instance, we cannot solve it in polynomial time by applying directly an algorithm for linear programming. However, we propose an alternative way for solving it similar to the one in Section 4.1, through its dual.

We associate to the constraints (4.3) and (4.4) the dual variables λ_j and μ_I, respectively. The dual LP of the relaxed LP is the following.

$$\max \sum_{J_j \in \mathcal{J}} \lambda_j - \sum_{I \in \mathcal{I}} \mu_I$$

$$\lambda_j - \sum_{I \in (i,j,c)} \mu_I \leq E_{i,j,c} \qquad P_i \in \mathcal{P},\ J_j \in \mathcal{J},\ c \in \mathcal{C}$$

$$\lambda_j, \mu_I \geq 0 \qquad J_j \in \mathcal{J},\ I \in \mathcal{I}$$

In order to solve this LP, we will show how to apply the Ellipsoid algorithm by constructing a polynomial-time separation oracle which runs in polynomial time. That is, given a solution to the dual LP, i.e. values to the variables λ_j and μ_I, we will define an algorithm which decides if the solution is feasible, and if not it identifies a violated constraint. At this point, recall our discussion in Section 4.1 on how we can solve an LP with an exponential number of constraints.

A polynomial-time separation oracle for the dual LP works as follows. Given a solution for the dual LP, for each job $J_j \in \mathcal{J}$, we want to find the minimum value $E_{i,j,c} + \sum_{I:(i,j,c) \in I} \mu_I$ among all configurations for J_j. Recall that a configuration is defined as the set of equal-length slots in which J_j is executed on a single processor $P_i \in \mathcal{P}$. Despite the fact that we have an exponential number of configurations, the number of possible distinct values of $E_{i,j,c}$ on the processor P_i is polynomial, as the slots are of equal length and hence the energy consumption depends only on the number of slots contained in each configuration.

Consider now the configurations of the job $J_j \in \mathcal{J}$ on a processor $P_i \in \mathcal{P}$ that contain exactly q slots. As the quantity $E_{i,j,c}$ is the same for all of these configurations, we want to find the configuration of q slots with the minimum $\sum_{I \in (i,j,c)} \mu_I$. Recall, that each slot consists of a subset of intervals of $\mathcal{I}$. Thus, we can compute for each slot t the quantity $\sum_{I \in t} \mu_I$. Therefore, we have just to select the q slots with the minimum values of $\sum_{I \in t} \mu_I$.

In total, for each job $J_j \in \mathcal{J}$ we can compute in polynomial time the configuration with the minimum $E_{i,j,c} + \sum_{I:(i,j,c)\in I} \mu_I$, among all processors and q's. If this quantity is less than λ_j then we have a violated constraint. Otherwise, if this quantity is greater than λ_j for all jobs, then the solution is feasible.

Therefore, there is a polynomial time separation oracle for the dual problem which runs in polynomial time and by applying the Ellipsoid algorithm we can compute efficiently an optimal solution for the dual program. Then, we can find an optimal solution for the relaxed primal LP by solving it with the variables corresponding to the constraints of the dual that were found to be violated during the run of the Ellipsoid algorithm to the dual and setting all the other primal variables equal to zero. The number of these constraints is polynomial to n and $1/\epsilon$. Thus, we can solve the relaxed primal LP with a polynomial number of variables and we can get an $((1+\frac{\epsilon}{1-\epsilon})(1+\frac{1}{n-2}))^{\alpha} \cdot OPT$ solution in polynomial time.

As we noticed in Section 4.1, in order to ensure that the Ellipsoid algorithm is polynomial we need to check two additional technical conditions. First, we have to show that the values of all the dual variables are upper bounded by some number R. For this, it suffices to argue as in Section 4.1. That is, the condition is satisfied because any optimal solution is a vertex of the corresponding polyhedron and we know that the value of the optimal solution is bounded. Subsequently, we have to show that there is a feasible point (or solution) for the dual LP and every point in a radius r is feasible. This is satisfied for the point (λ, μ) defined as follows. We set $\lambda_j = 0$, for all $J_j \in \mathcal{J}$, $\mu_I = 0$ for all $I \in \mathcal{I}$ and $r = \min_{i,j,c} E_{i,j,c}$. Thus, we can indeed solve the dual LP in polynomial time with the Ellipsoid algorithm.

Technical Lemmas

Before presenting our algorithm, we state and prove some technical lemmas that we need for its analysis. Lemma 4.3 deals with the expressions arising when one estimates the moments of random variables with Binomial distributions. Lemma 4.4 estimates moments of Binomial random variables through the moments of Poisson random variables. Finally, Lemma 4.6 estimates moments of Poisson random variables with parameter $\lambda < 1$ through the moments of Poisson random variables with parameter 1.

Lemma 4.3. *Consider a set of real numbers $\{X_1, X_2, \ldots, X_n\}$ such that $X_j \in [0,1]$ for all $j \in \{1,\ldots,n\}$ and a set of non-negative constants $\{e_1, e_2, \ldots, e_n\}$. Assume that we split X_n to $X'_n \geq 0$ and $X'_{n+1} \geq 0$ so that $X_n = X'_n + X'_{n+1}$. Let $X'_j = X_j$, for $j \in \{1,2,\ldots,n-1\}$, and $e_{n+1} = e_n$. Then, it holds that*

$$\sum_{S\subseteq\{1,2,\ldots,n\}} |S|^{\alpha-1} \left(\sum_{j\in S} e_j\right) \prod_{j\in S} X_j \prod_{j\notin S}(1-X_j) \leq \sum_{S\subseteq\{1,2,\ldots,n+1\}} |S|^{\alpha-1} \left(\sum_{j\in S} e_j\right) \prod_{j\in S} X'_j \prod_{j\notin S}(1-X'_j)$$

Proof. The left-hand side of the inequality of the statement can be rewritten as

$$\sum_{S'\subseteq\{1,2,\ldots,n-1\}} \prod_{j\in S'} X_j \prod_{j\in\{1,2,\ldots,n-1\}\setminus S'} (1-X_j)$$
$$\times \left[(1-X_n)|S'|^{\alpha-1} \sum_{j\in S'} e_j + X_n(|S'|+1)^{\alpha-1} \sum_{j\in S'\cup\{n\}} e_j\right] \tag{4.6}$$

and the right-hand side as

$$\sum_{S'\subseteq\{1,2,\ldots,n-1\}} \prod_{j\in S'} X'_j \prod_{j\in\{1,2,\ldots,n-1\}\setminus S'} (1-X'_j)$$
$$\times \left[(1-X'_n)(1-X'_{n+1})|S'|^{\alpha-1}\sum_{j\in S'} e_j + X'_n(1-X'_{n+1})(|S'|+1)^{\alpha-1}\sum_{j\in S'\cup\{n\}} e_j \right.$$
$$\left. +(1-X'_n)X'_{n+1}(|S'|+1)^{\alpha-1}\sum_{j\in S'\cup\{n+1\}} e_j + X'_nX'_{n+1}(|S'|+2)^{\alpha-1}\sum_{j\in S'\cup\{n,n+1\}} e_j\right] \tag{4.7}$$

As $X'_j = X_j$, for $1 \le j \le n-1$, it suffices to compare only the terms inside the brackets of (4.6) and (4.7). For (4.6), let $c = 1 - X_n$, $A = |S'|^{\alpha-1}(\sum_{j\in S'} e_j)$ and $A' = (|S'|+1)^{\alpha-1}(\sum_{j\in S'\cup\{n\}} e_j)$. Then, we can write the term inside the brackets of (4.6) as

$$cA + (1-c)A'$$

where $A < A'$. For (4.7), let $c_1 = (1-X'_n)(1-X'_{n+1})$, $c_2 = X'_n(1-X'_{n+1})$, $c_3 = (1-X'_n)X'_{n+1}$ and $c_4 = X'_nX'_{n+1}$. Note that $c_1+c_2+c_3+c_4 = 1$ and $c_1 < c$. As before, $A = |S'|^{\alpha-1}(\sum_{j\in S'} e_j)$ and $A' = (|S'|+1)^{\alpha-1}(\sum_{j\in S'\cup\{n\}} e_j)$. Moreover, let $A'' = (|S'|+2)^{\alpha-1}(\sum_{j\in S'\cup\{n,n+1\}} e_j)$. So, we can write the term inside the brackets of (4.7) as

$$c_1A + c_2A' + c_3A' + c_4A'' \tag{4.8}$$

where $A' < A''$. Since $A' < A''$ and $c_1+c_2+c_3+c_4 = 1$, we get that

$$\begin{aligned} c_1A + c_2A' + c_3A' + c_4A'' &> c_1A + (c_2+c_3+c_4)A' \\ &= c_1A + (1-c_1)A' \\ &> cA + (1-c)A' \end{aligned}$$

and the lemma follows. □

Lemma 4.4. *For any $\alpha \ge 1$, the function $f(x) = x^\alpha$ and a parameter $a \in [0,1]$ we have*

$$\mathbb{E}[f(B_a)] \le \mathbb{E}[f(P_a)]$$

where B_a is a sum of n independent Bernoulli random variables with expected value $\mathbb{E}[B_a] = a$ and P_a is a Poisson random variable with parameter a.

Proof. To upper bound the expected value of $f(x)$, we will need the following probabilistic fact that was first proved by Hoeffding [44] for a finite sum of Bernoulli random variables and was lately generalized for more general distributions by Berend et al. [22].

Lemma 4.5 ([22])**.** *Let $X = \sum_{i=1}^t X_i$ be the sum of t (where t is possibly equal to infinity) independent random variables, $0 \le X_i \le 1$ for $i = 1,\ldots,t$ and $\mu = \mathbb{E}[X]$. For every convex function f,*

$$\mathbb{E}[f(X)] \le \mathbb{E}[f(Y)]$$

where Y is a binomial random variable with distribution $Y \sim B(t, \mu/t)$ in case $t < \infty$, and a Poisson random variable with distribution $Y \sim P(\mu)$ otherwise.

We define a binomial random variable B'_a as a sum of B_a and an infinite number of Bernoulli random variables X'_i for $i = 1, \ldots, \infty$ such that $Pr(X_i = 1) = 0$. Obviously, $\mathbb{E}[B'_a] = \mathbb{E}[B_a] = a$ and $\mathbb{E}[f(B'_a)] = \mathbb{E}[f(B_a)]$. Since the function $f(x)$ is convex we can apply the Lemma 4.5 with parameter $t = \infty$, and the statement follows. □

Lemma 4.6. *Consider any real number $\alpha \geq 1$ and a Poisson random variable P_λ with parameter $0 \leq \lambda \leq 1$. It holds that $\mathbb{E}[(P_\lambda)^\alpha] \leq \lambda \mathbb{E}[(P_1)^\alpha]$.*

Proof. Note that $\mathbb{E}[(P_\lambda)^\alpha] = \sum_{k=0}^{\infty} k^\alpha \frac{\lambda^k e^{-\lambda}}{k!}$. Moreover, $e^{-(1-\lambda)} \geq 1 - (1-\lambda) = \lambda \geq \lambda^{k-1}$ for $k \geq 2$ and $0 \leq \lambda \leq 1$. Therefore, $e^{-1} \geq \lambda^{k-1} e^{-\lambda}$ for all $k \geq 2$. For $\lambda = 0$, the statement of the Lemma is trivial. Assume that $\lambda > 0$. Then,

$$\begin{aligned}
\mathbb{E}[(P_1)^\alpha] - \frac{1}{\lambda}\mathbb{E}[(P_\lambda)^\alpha] &= \sum_{k=0}^{\infty} k^\alpha \frac{e^{-1} - \lambda^{k-1} e^{-\lambda}}{k!} \\
&= (e^{-1} - e^{-\lambda}) + \sum_{k=2}^{\infty} k^\alpha \frac{e^{-1} - \lambda^{k-1} e^{-\lambda}}{k!} \\
&\geq (e^{-1} - e^{-\lambda}) + \sum_{k=2}^{\infty} k \frac{e^{-1} - \lambda^{k-1} e^{-\lambda}}{k!} \\
&= \sum_{k=0}^{\infty} k \frac{e^{-1}}{k!} - \frac{1}{\lambda} \sum_{k=0}^{\infty} k \frac{\lambda^k e^{-\lambda}}{k!} \\
&= 1 - \frac{1}{\lambda} \cdot \lambda = 0.
\end{aligned}$$

□

Randomized Rounding Algorithm

Next, we elaborate on our approximation algorithm which constructs a feasible schedule for our scheduling problem.

Previously, we showed that we can compute an optimal solution for the fractional relaxation of the integer configuration LP in polynomial time. Unfortunately, this solution is not feasible for the integer LP and it does not correspond to a feasible schedule of our scheduling problem. Notice, however, that any feasible solution $\tilde{x}$ of the relaxed LP can be interpreted as *fractional schedule*. Let $|I|$ be the length of the interval $I \in \mathcal{I}$. Then, every variable $\tilde{x}_{i,j,c} > 0$ can be interpreted as a set of rectangular pieces, one for each $I \in (i, j, c)$. Each of these rectangular pieces has length $|I|$ and height $s_{i,j,c}$, where $s_{i,j,c}$ is the speed of the job J_j if it is entirely executed on the processor P_i according to the configuration c. Given this interpretation, the basic reason why the fractional solution is not feasible is because a processor might execute more than one jobs at the same time in the fractional schedule. Note that we can turn the fractional schedule into a feasible schedule for our problem in the following manner. If K_I jobs are executed during the interval I in the fractional schedule, we simply have to increase the speeds of these jobs by a factor of K_I during I and schedule the jobs in a feasible manner during each interval I. For example, see Figure 4.2.

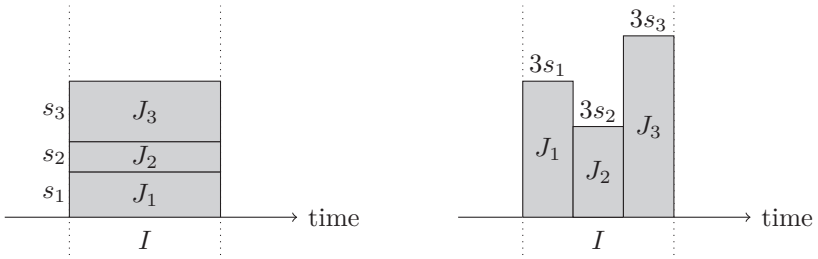


Figure 4.2: An example of a fractional schedule in which three jobs are executed during the interval I. That is, the only positive variables $\tilde{x}_{i,j,c}$ are the ones which correspond to the jobs J_1, J_2 and J_3. By increasing their speeds by a factor of three during I, we can turn the fractional schedule into a feasible one for our original problem.

We are now ready to describe our algorithm. First, it solves the fractional relaxation of the integer configuration LP. Subsequently, it applies randomized rounding in order to choose a configuration for each job. Then, it is possible that the schedule is not feasible because more than one jobs might have to be executed at the same time by the same processor. So the algorithm scales the speeds of the jobs in order to produce a feasible schedule. See Algorithm 4.1.

Algorithm 4.1

1: Compute an optimal solution $\tilde{x}$ for the fractional relaxation of the integer LP.
2: Schedule J_j on P_i according to the configuration c with probability $\tilde{x}_{i,j,c}$.
3: Let K_I be the number of selected configurations that contain the interval I. Scale the speed of every job executed during I by a factor of K_I.

Theorem 4.2. *Algorithm 4.1 achieves an approximation ratio of* $((1+\frac{\varepsilon}{1-\varepsilon})(1+\frac{2}{n-2}))^{\alpha}\tilde{B}_{\alpha_{max}}$ *for the heterogeneous multiprocessor preemptive speed-scaling problem without migrations in time polynomial to the size of the instance and* $1/\varepsilon$*, where* $\alpha_{max} = \max_{P_i \in \mathcal{P}} \alpha_i$.

Proof. For each interval $I \in \mathcal{I}$ on every processor, we estimate its expected energy consumption. So, consider an interval I for a processor P_i. Let X_j be the probability that the job J_j is assigned to be processed during the interval I by the randomized rounding. We have that

$$X_j = \sum_{P_i \in \mathcal{P}} \sum_{c \in \mathcal{C}} \tilde{x}_{i,j,c}$$

By the constraint (4.4), we know that $\sum_{J_j \in \mathcal{J}} X_j \leq 1$. The expected energy that the job J_j consumes during the interval I under the condition that J_j is assigned to be processed in the interval I without considering the other jobs is

$$E_j = \sum_{P_i \in \mathcal{P}} \sum_{c \in \mathcal{C}_j} \frac{|I| s_{i,j,c}^{\alpha_i} \tilde{x}_{i,j,c}}{X_j}$$

The energy consumption during the interval I achieved by the optimal fractional solution of the relaxed LP is

$$LP^* = \sum_{J_j \in \mathcal{J}} E_j \cdot X_j$$

If the randomized rounding assigns the set S of jobs to be processed during the interval I, then we need to speed up the execution of all jobs in the interval I by a factor of $|S|$. This means that the energy consumption increases by the factor $|S|^{\alpha_i - 1}$. Therefore, during the interval I, the expected energy consumption in the final schedule is

$$E = \sum_{S \subseteq \mathcal{J}} |S|^{\alpha_i - 1} \left(\sum_{J_j \in S} E_j \right) Pr(S)$$

where $Pr(S)$ is the probability that exactly the jobs in the set S are selected during I. Therefore, we have that

$$E = \sum_{S \subseteq \mathcal{J}} |S|^{\alpha_i - 1} \left(\sum_{J_j \in S} E_j \right) \prod_{J_j \in S} X_j \prod_{J_j \in \mathcal{J} \setminus S} (1 - X_j)$$

We can assume that there exists a $Q \in \mathbb{N}$ such that $X_j = \frac{q_j}{Q}$, $J_j \in \mathcal{J}$, for some $q_j \in \mathbb{N}$ since the numbers X_j come from solving an LP with rational coefficients. Note that we do not make any assumptions on the encoding length of these numbers and we use them only for analysis purposes. Clearly, $q_j \leq Q$ for every $J_j \in \mathcal{J}$, since $X_j \leq 1$. Hence, we can chop each X_j into q_j pieces $X_{j,1}, X_{j,2}, \ldots, X_{j,q_j}$ such that $X_{j,\ell} = \frac{1}{Q} = X$, for $1 \leq \ell \leq q_j$. Let $q = \sum_{j=1}^{n} q_j$ be the number of all chopped pieces and $e_{j,\ell} = E_j$, for $1 \leq j \leq n$ and $1 \leq \ell \leq q_j$. Note that, $q \leq Q$ since $\sum_{j=1}^{n} X_j \leq 1$. For the ease of exposition we identify the set $\{1, 2, \ldots, q\}$ with the set of all pairs (j, ℓ) such that $1 \leq j \leq n$ and $1 \leq \ell \leq q_j$. By using Lemma 4.3 we get

$$\begin{aligned} E &\leq \sum_{S \subseteq \{1,2,\ldots,q\}} |S|^{\alpha_i - 1} \left(\sum_{(j,\ell) \in S} e_{j,\ell} \right) X^{|S|} (1 - X)^{q - |S|} \\ &= \sum_{k=1}^{q} \sum_{S \subseteq \{1,2,\ldots,q\}, |S| = k} \left(\sum_{(j,\ell) \in S} e_{j,\ell} \right) k^{\alpha_i - 1} X^k (1 - X)^{q-k} \end{aligned}$$

By changing the order of the sums in the above inequality we get

$$\begin{aligned} E &\leq \left(\sum_{j=1}^{n}\sum_{\ell=1}^{q_j} e_{j,\ell}\right)\sum_{k=1}^{q}\binom{q-1}{k-1}k^{\alpha_i-1}X^k(1-X)^{q-k} \\ &= \left(\sum_{J_j\in\mathcal{J}} q_jE_j\right)\sum_{k=1}^{q}\binom{q}{k}\frac{\binom{q-1}{k-1}}{\binom{q}{k}}k^{\alpha_i-1}X^k(1-X)^{q-k} \\ &= \left(\frac{1}{q}\sum_{J_j\in\mathcal{J}} q_jE_j\right)\sum_{k=1}^{q}\binom{q}{k}k^{\alpha_i}X^k(1-X)^{q-k} \\ &= \frac{Q}{q}LP^*\sum_{k=1}^{q}\binom{q}{k}k^{\alpha_i}X^k(1-X)^{q-k} \\ &\leq \frac{Q}{q}LP^*\mathbb{E}[(B_{q/Q})^{\alpha_i}] \end{aligned}$$

where $\binom{q-1}{k-1}$ is the number of sets of cardinality k that contain J_j. Moreover, $B_{q/Q}$ is a random variable with expectation $\frac{q}{Q}$ which corresponds to the sum of q independent Bernoulli random variables. Therefore,

$$E \leq \frac{Q}{q}LP^*\cdot\mathbb{E}[(B_{q/Q})^{\alpha_i}] \leq \frac{Q}{q}LP^*\cdot\mathbb{E}[(P_{q/Q})^{\alpha_i}] \leq \frac{Q}{q}LP^*\cdot\frac{q}{Q}\mathbb{E}[(P_1)^{\alpha_i}]$$

where the second inequality follows from Lemma 4.4 and the last inequality follows from Lemma 4.6. Therefore, by summing over all intervals and processors and as $\alpha_{max} = \max_{i\in\mathcal{P}}\alpha_i$, we get

$$E \leq LP^*\cdot\mathbb{E}[(P_1)^{\alpha_{max}}] = LP^*\cdot\tilde{B}_{\alpha_{max}}$$

□

Compact Linear Programming Relaxation

Before, we showed a way of solving the fractional relaxation of the integer configuration LP for our problem by applying the Ellipsoid algorithm. Subsequently, we present another way of solving this LP by using as a black box any algorithm for linear programming. Therefore, we can obtain an optimal fractional solution by using a faster algorithm instead of the Ellipsoid algorithm.

Our approach is the following. We formulate the problem as a compact integer LP with a polynomial number of contraints and we show that the fractional relaxation of this LP is equivalent with the relaxed configuration LP. Specifically, we show that, given an optimal fractional solution of the compact LP, we can obtain an optimal fractional solution for the configuration LP in polynomial time.

In the following we define a compact formulation for the problem without migrations and we show that the relaxations of the compact and the configuration LPs are equivalent. Recall that, by Lemma 4.2, there is always an $((1+\frac{\varepsilon}{1-\varepsilon})(1+\frac{2}{n-2}))^{\alpha}$-approximate schedule for our problem such that if the job $J_j \in \mathcal{J}$ is executed on the processor $P_i \in \mathcal{P}$, then its feasibility interval $[r_{i,j}, d_{i,j})$ can be partitioned into equal-length slots. Given such a slot t, J_j is either executed during the whole t or it is not executed at all during t. The

number of these slots is $\frac{n^3}{\varepsilon}$, while each slot t has length $\ell_t = \frac{\varepsilon}{n^3}(d_{i,j} - r_{i,j})$. Recall also that $\mathcal{I}$ denotes the set of all intervals occurred by merging the slots for all jobs.

In order to formulate our problem as a compact LP, we introduce a binary variable $y_{i,j,q}$ which is equal to one if the job J_j is executed on the processor P_i during exactly q slots and zero otherwise. Moreover, we introduce a binary variable $z_{i,j,q,t}$ which is equal to one if the job J_j is executed on the processor P_i during the slot t and it is executed during exactly q slots in total. Otherwise, $z_{i,j,q,t}$ is equal to zero. We define the constants $p_{i,j,q} = q\frac{\varepsilon}{n^3}(d_{i,j} - r_{i,j})$ and $E_{i,j,q} = \frac{w_{i,j}^{\alpha_i}}{p_{i,j,q}^{\alpha_i - 1}}$. Clearly, $p_{i,j,q}$ and $E_{i,j,q}$ correspond to the total execution time and the energy consumption, respectively, of the job J_j if it is entirely executed on the processor P_i during exactly q slots. Then, our problem can be formulated as follows.

$$\min \sum_{P_i \in \mathcal{P}} \sum_{J_j \in \mathcal{J}} \sum_{q=1}^{n^3/\epsilon} E_{i,j,q} \cdot y_{i,j,q}$$

$$\sum_{P_i \in \mathcal{P}} \sum_{q=1}^{n^3/\epsilon} y_{i,j,q} = 1 \qquad J_j \in \mathcal{J} \tag{4.9}$$

$$\sum_{t=1}^{n^3/\epsilon} z_{i,j,q,t} = q \cdot y_{i,j,q} \qquad P_i \in \mathcal{P}, J_j \in \mathcal{J}, q \in \{1, 2, \ldots, \tfrac{n^3}{\varepsilon}\} \tag{4.10}$$

$$\sum_{J_j \in \mathcal{J}} \sum_{q=1}^{n^3/\epsilon} \sum_{t: I \subseteq t} z_{i,j,q,t} \leq 1 \qquad P_i \in \mathcal{P}, I \in \mathcal{I} \tag{4.11}$$

$$y_{i,j,q}, z_{i,j,q,t} \in \{0, 1\} \qquad P_i \in \mathcal{P}, J_j \in \mathcal{J}, q, t \in \{1, 2, \ldots, \tfrac{n^3}{\varepsilon}\} \tag{4.12}$$

The constraints (4.9) ensure that each job is entirely executed on some processor. The constraints (4.10) establish the relationship between the variables $z_{i,j,q,t}$ and $y_{i,j,q}$. If $y_{i,j,q} = 1$, then exactly q variables $z_{i,j,q,t}$ must be equal to one. The constraint (4.11) enforces that at most one job is executed by each processor at each time. Recall that, given a job $J_j \in \mathcal{J}$ which is executed on the processor $P_i \in \mathcal{P}$, if J_j is executed during the slot $t \in \{1, 2, \ldots, \frac{n^3}{\varepsilon}\}$, then J_j is executed during every interval $I \in \mathcal{I}$ such that $I \subseteq t$. Note that the numbers of both the variables and the constraints of the above LP are polynomial to n and $1/\varepsilon$.

The configuration and the compact formulations are equivalent, as they both lead to a minimum energy schedule satisfying Lemma 4.2. Consider now the LP's that occur if we relax constraints (4.5) and (4.12), respectively. In Lemma 4.7 we prove that the equivalence is also true for these relaxations, through a transformation of a solution for the relaxed configuration LP to a solution for the relaxed compact LP of the same energy consumption, and vice versa. As a result, given a solution of the relaxed compact LP obtained by any polynomial time algorithm, we can get a solution for the relaxed configuration LP in polynomial time. Then, we can apply the randomized rounding presented in the previous section and get the approximation ratio of Theorem 4.2.

Lemma 4.7. *The relaxations of the configuration LP and the compact LP are equivalent.*

Proof. We will show that any feasible solution for the relaxed configuration LP can be

transformed to a feasible solution for the relaxed compact LP of the same energy consumption and vice versa.

Assume that we are given a feasible solution for the relaxation of the configuration LP. Such a solution corresponds to a schedule of the jobs on the processors. Specifically, the value of the variable $x_{i,j,c}$ specifies the part of the job $J_j \in \mathcal{J}$ executed on processor $P_i \in \mathcal{P}$ during the slots that belong to the configuration $c \in \mathcal{C}$. Then, we define $z_{i,j,q,t} = \sum_{c \in \mathcal{C}: t \in c} x_{i,j,c}$. This defines a feasible solution for the relaxation of the compact LP with the same energy consumption.

Assume that we are given a feasible solution for the compact LP. We will define a set of configurations and we will assign a non-zero value for each variable $x_{i,j,c}$ that corresponds to these configurations. The number of these configurations should be polynomial to n and $\frac{1}{\varepsilon}$. The remaining variables of the configuration LP will be set to zero.

Consider a non-zero variable $y_{i,j,q}$ (and its corresponding variables $z_{i,j,q,t}$) in the solution of the compact LP. We partition the part of the schedule defined by $y_{i,j,q}$ into a set of configurations with q slots and we specify the values of the variables $x_{i,j,c}$ that correspond to these configurations. To do this, for each variable $y_{i,j,q}$ and its associated variables $z_{i,j,q,t}$, we construct a bipartite graph $G = (A \cup B, E)$ as follows. The set A contains q nodes, i.e. $A = \{a_1, a_2, \ldots, a_q\}$. Intuitively, each of these nodes corresponds to one of the q slots of the configurations that will correspond to $y_{i,j,q}$. The set B contains $\frac{n^3}{\varepsilon}$ nodes, one for each possible slot of J_j on the processor P_i (see Lemma 4.2), i.e. $B = \{b_1, b_2, \ldots, b_{\frac{n^3}{\varepsilon}}\}$. We will define the set of edges E and their weights, such that each node $a_k \in A$ has weighted degree exactly $y_{i,j,q}$ and each node $b_t \in B$ has weighted degree exactly $z_{i,j,q,t}$. Note that, the total weight of all the edges will be $q \cdot y_{i,j,q} = \sum_t z_{i,j,q,t}$. We start by adding edges from a_1 to $b_1, b_2, \ldots$ of weight $z_{i,j,q,1}, z_{i,j,q,2}, \ldots$, respectively, as long as $\sum_{t=1}^{k} z_{i,j,q,t} \leq y_{i,j,q}$. The first time where $\sum_{t=1}^{k} z_{i,j,q,t} > y_{i,j,q}$ we add an edge between a_1 and b_k of weight $y_{i,j,q} - \sum_{t=1}^{k-1} z_{i,j,q,t}$. Moreover, we add an edge between a_2 and b_k of weight $z_{i,j,q,k} - (y_{i,j,q} - \sum_{\ell=1}^{k-1} z_{i,j,q,t})$. We continue adding edges from a_2 to $b_{k+1}, b_{k+2}, \ldots$ of weight $z_{i,j,q,k+1}, z_{i,j,q,k+2}, \ldots$, respectively, until the sum of their weights is bigger than $y_{i,j,q}$. At this point we add an edge of appropriate weight. Then, we start from a_3 and we continue like this. Note that, by construction each node $b_t \in B$ has degree either one or two.

Consider now the weighted graph G' that occurs from G if all edge weights are divided by $y_{i,j,q}$. In G', the weighted degree of each node $a_k \in A$ is exactly one while the weighted degree of each node $b_t \in B$ is at most one. The following lemma follows directly from the integrality of the bipartite perfect matching polytope (see [58] for a thorough discussion on the topic).

Proposition 4.1. *Let $G = (A \cup B, E)$ be a bipartite graph in which each node in A has weighted degree exactly one and each node in B has weighted degree at most one. There are perfect matchings $M_1, M_2, \ldots, M_r$ (i.e., matchings having exactly $|A|$ edges) and coefficients $\lambda_1, \lambda_2, \ldots, \lambda_r$ such that $\sum_{i=1}^{r} \lambda_i = 1$, and for each edge e it holds that $\sum_{i: e \in M_i} \lambda_i = w_e$, where w_e is the weight of the edge e.*

Note that each matching in G' corresponds to a feasible configuration for the job J_j. Hence, applying the above proposition to G', we get a set of r configurations for J_j. Note

that r is at most the number of the edges of G' which is polynomial to n and to $1/\varepsilon$. For each configuration c that corresponds to the matching M_c, we set $x_{i,j,c} = \lambda_c \cdot y_{i,j,q}$.

It is easy to see that the solution obtained for the configuration LP is feasible. The fact that constraints (4.3) are satisfied comes from constraints (4.9) and (4.10) while the constraints (4.4) are satisfied due to the constraints (4.11). □

4.3 Average Completion Time Plus Energy Minimization

In this section, we consider the problem $\mathbf{S, R|w_{i,j}|\sum C_j + \beta E}$ of minimizing a linear combination of the sum of completion times of a set of jobs and their total energy consumption on parallel heterogeneous processors and we propose an optimal polynomial algorithm which is based on a formulation of the problem as a minimum weighted maximum matching problem.

In $\mathbf{S, R|w_{i,j}|\sum C_j + \beta E}$, there is a set of n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ which have to be scheduled on a set of m parallel heterogeneous processors $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$. The fact that the processors are heterogeneous means that each processor $P_i \in \mathcal{P}$ satisfies its own speed-to-power function $Q_i = s^{\alpha_i}$. In this problem, we do not allow preemptions and migrations of jobs. That is, each job has to be executed on a single processor without interruptions. Each job $J_j \in \mathcal{J}$ has an amount of work $w_{i,j}$ to accomplish if it is executed on the processor $P_i \in \mathcal{P}$. All the jobs are released at the time $t = 0$. The goal is to minimize the sum of completion times of all the jobs plus β times their total energy consumption. The parameter $\beta > 0$ is used to specify the relevant importance of the average completion time criterion versus the total energy consumption criterion.

For $\mathbf{S, R|w_{i,j}|\sum C_j + \beta E}$, we propose an optimal polynomial time algorithm. The main idea of our algorithm is to formulate this problem as a minimum weighted maximum matching problem on an appropriate bipartite graph. This formulation is based on three observations. Firstly, based on the convexity of the speed-to-power function of each processor, we can show that there is always an optimal schedule for the problem such that each job $J_j \in \mathcal{J}$ is executed with constant speed and there is no idle time on any processor $P_i \in \mathcal{P}$ until the last job on P_i completes. Secondly, the fact that preemption and migration of jobs is not allowed means that there is an order of the jobs executed by any processor in any feasible schedule. Given such a schedule, if ℓ jobs are executed by the processor $P_i \in \mathcal{P}$, then we can consider that there are ℓ available positions on P_i, one for the execution of each of these ℓ jobs. If the job J_j is executed in the k-th position of the processor P_i, then $k-1$ jobs precede J_j and $\ell - k$ jobs succeed J_j. Clearly, there can be at most n such positions for each processor. Finally, the contribution of a job J_j to the objective function depends only on its position on the processor by which it is executed and it is independent of where the other jobs are executed. Overall, our problem reduces to assigning every job to a position of a processor so that our objective is minimized.

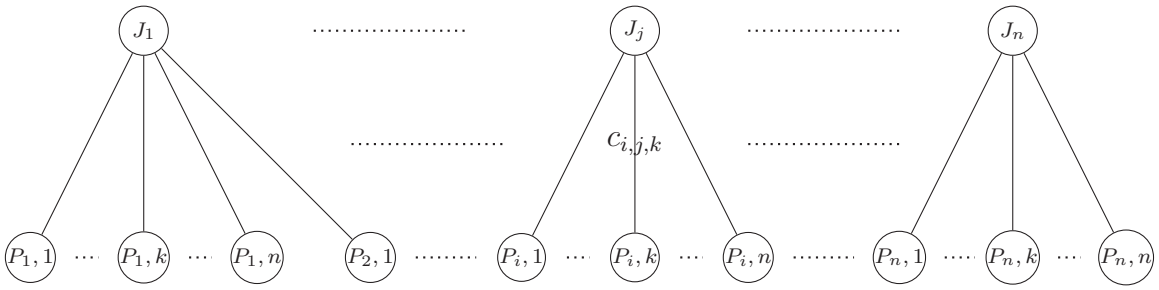
In order to formulate $\mathbf{S, R|w_{i,j}|\sum C_j + \beta E}$ as a minimum weighted maximum matching problem, we define a bipartite graph G whose edges are weighted. The following lemma is our guide for assigning weights to the edges of G and fixes the cost, i.e. the contribution to the objective function, of executing a job J_j to the k-th position of any

processor.

Lemma 4.8. *Assume that, in an optimal schedule for* $\mathbf{S,R|w_{i,j}|\sum C_j + \beta E}$*, the job* $J_j \in \mathcal{J}$ *is executed with speed* s_j *on processor* P_i *in the* k*-th position from the end of* P_i*. Then, the contribution of* J_j *to the objective function is minimized if it holds that* $s_j = (\frac{k}{(\alpha_i - 1)\beta})^{1/\alpha_i}$.

Proof. Let s_j be the speed of J_j. As J_j is executed on the processor P_i, its processing time is $\frac{w_{i,j}}{s_j}$. Since $k-1$ jobs follow J_j on P_i, the term $\frac{w_{i,j}}{s_j}$ is added k times on the sum of completion times of all the jobs. Moreover, $w_{i,j} \cdot s_j^{\alpha_i - 1}$ units of energy are consumed for the execution of the job J_j. Hence, the total contribution of J_j to the objective is $k \cdot \frac{w_{i,j}}{s_j} + \beta \cdot w_{i,j} \cdot s_j^{\alpha_i - 1}$. By differentiating the last term with respect to s_j and setting this derivative equal to zero, we can get the value of s_j for which this contribution is minimized. □

The above lemma specifies the speed that the job J_j must have in an optimal schedule, if it is executed on the k-th position of the processor P_i. In the following, we denote this speed as $s^*_{i,j,k}$. In order to formulate our problem as a minimum weighted maximum matching, we create the complete bipartite graph $G = (V \cup U, A)$ as follows: (i) for each job $J_j \in \mathcal{J}$, we add a vertex in V, (ii) for every pair of processor $P_i \in \mathcal{P}$, and position k, $1 \leq k \leq n$, (counting from the end) we add a vertex in U, and (iii) for each edge $(J_j, (P_i, k)) \in A$, we set its weight $c_{i,j,k} = k \cdot \frac{w_j}{s^*_{i,j,k}} + \beta \cdot w_{i,j} \cdot (s^*_{i,j,k})^{\alpha_i - 1}$.



A description of our algorithm follows.

Algorithm 4.2

1: Construct the bipartite graph G.
2: Find a minimum weighted maximum matching M in G.
3: **for** each $(J_j, (M_i, k)) \in M$ **do**
4: Schedule J_j to the position k of P_i with speed $s^*_{i,j,k}$.

Theorem 4.3. *Algorithm 4.2 is optimal for* $\mathbf{S,R|w_{i,j}|\sum C_j + \beta E}$.

Proof. By the construction of G, the vertex $J_j \in \mathcal{J}$ can belong to at most one edge of the matching. Moreover, the number of the job nodes is less than the number of the processor-position nodes and every job node is connected with every processor-position node. Hence, every job node belongs to a maximum matching of G. Therefore, each job is scheduled on a single processor and, as a result, the schedule produced by the algorithm is feasible.

We next prove the optimality of our algorithm. Among the matchings with cardinality n, the algorithm finds the one so that the total contribution of each job $J_j \in \mathcal{J}$ to the objective of minimizing the average completion time plus energy is minimized. Note that the speed s_j of the job J_j is selected in an optimal way according to Lemma 4.8. In other words, given the construction of the bipartite graph G, the algorithm finds the schedule with the minimum average completion time plus energy. Hence, our algorithm is optimal. □

Chapter 5

Shop Environments

In this chapter, we consider speed scaling problems in shop environments.

Initially, in Section 5.1, we address the problem $\mathbf{S}, \mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|\mathbf{E}$ of minimizing the energy consumption in an open shop. For this problem, we present two optimal algorithms. Firstly, we construct an optimal algorithm which is based on a primal-dual schema. Unfortunately, we do not know how to compute the worst-case running time of this algorithm. So, we evaluate its performance experimentally. Our experiments indicate that, in general, the algorithm's running time is linear with the number of the jobs. However, in the very specific case where the number of the jobs is equal to the number of the processors, there is a burst in its running time. We also compare the execution of the primal-dual algorithm with a commercial solver. Subsequently, we describe an optimal polynomial-time algorithm for the open shop problem which is based on a formulation as a convex cost flow problem.

Next, in Section 5.2, we study the energy minimization problem $\mathbf{S}, \mathbf{J}|\mathbf{r_{i,j}}, \mathbf{d_{i,j}}, \mathbf{pmtn}|\mathbf{E}$ in a job shop environment and we propose a $(1+\epsilon)\tilde{B}_{\alpha_{max}}$-approximation algorithm, where $\tilde{B}_{\alpha_{max}}$ is the α_{max}-th generalized Bell number. First, we formulate the problem as an integer configuration linear program. Then we give an algorithm which solves its fractional relaxation and applies randomized rounding in order to compute a feasible schedule.

5.1 Energy Minimization in an Open Shop

In this section, we study the energy minimization problem $\mathbf{S}, \mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|\mathbf{E}$ in an open shop environment. In this problem, there is a set of n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ and a set of m processors $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$. Each job $J_j \in \mathcal{J}$ consists of m operations $O_{1,j}, O_{2,j}, \ldots, O_{m,j}$. For every operation $O_{i,j}$, $P_i \in \mathcal{P}$ and $J_j \in \mathcal{J}$, we are given an amount of work $w_{i,j} \geq 0$. The open shop constraint enforces that no two operations of the same job can be executed at the same time. Each operation $O_{i,j}$ must be entirely executed by the processor P_i. That is, each job $J_j \in \mathcal{J}$ has exactly one operation $O_{i,j}$ on each processor $P_i \in \mathcal{P}$. Obviously, if $w_{i,j} = 0$, then the processor P_i does not have to execute anything for the job J_j. In this setting, we allow preemptions of the operations, i.e. an operation may be executed, suspended and resumed later from the point of suspension. The objective is to find a minimum energy schedule such that all the operations are

executed during the interval $[0, d)$.

5.1.1 Optimal Primal-Dual Algorithm

In the following, we present an optimal algorithm for $\mathbf{S}, \mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|\mathbf{E}$ which is based on a primal-dual schema. Initially, we propose a convex programming formulation for the problem. Then, we associate to each constraint of this formulation a dual variable and we apply the KKT conditions which relate the primal variables to the dual variables with a set of equalities. Next, we define a primal-dual algorithm which produces an optimal solution for the convex program by properly modifying the dual variables. Note that a modification of the dual variables has a direct impact on the primal variables because of the KKT conditions. By solving the convex program we obtain a set of optimal speeds, i.e. processing times, for all the operations. If the operation $O_{i,j}$ is assigned the speed $s_{i,j}$, then its processing time is equal to $\frac{w_{i,j}}{s_{i,j}}$. Finally, we apply an optimal algorithm for the feasibility scheduling problem $\mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|-$ in order to produce the final optimal schedule.

In the problem $\mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|-$, we are given a set of n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ and a set of m processors $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$. Each job $J_j \in \mathcal{J}$ consists of a set of m operations as it is the case for $\mathbf{S}, \mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|\mathbf{E}$. The operation $O_{i,j}$, $P_i \in \mathcal{P}$ and $J_j \in \mathcal{J}$, must be entirely executed by the processor P_i and it has a processing time $p_{i,j} \geq 0$. We are constrained not to execute any pair of operations of a job at the same time. The goal is to find a feasible schedule such that all operations are scheduled preemptively during the interval $[0, d)$ or decide that such a schedule does not exist. This problem is polynomially solvable and an optimal algorithm can be found in [36].

Let us, now, give a convex programming formulation for $\mathbf{S}, \mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|\mathbf{E}$. Our formulation is based on the fact that there is always an optimal schedule in which each operation $O_{i,j}$ runs at a constant speed $s_{i,j}$, which comes from the convexity of the speed-to-power function. In order to establish a convex program, we also need some necessary and sufficient properties for the existence of a feasible schedule when we know the processing times of the operations. The following lemma describes such properties. Its proof is omitted and it can be found in [36].

Lemma 5.1. *An instance of* $\mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|-$ *is feasible if and only if*

- $\sum_{J_j \in \mathcal{J}} p_{i,j} \leq d$ *for all* $P_i \in \mathcal{P}$, *and*
- $\sum_{P_i \in \mathcal{P}} p_{i,j} \leq d$ *for all* $J_j \in \mathcal{J}$.

For notational convenience, we say that $O_{i,j} \in J_j$ and $O_{i,j} \in P_i$ if $w_{i,j} > 0$. We ignore any operation $O_{i,j}$ with $w_{i,j} = 0$. Moreover, let $\mathcal{O}$ be the set of all the operations $O_{i,j}$, $P_i \in \mathcal{P}$ and $J_j \in \mathcal{J}$, such that $w_{i,j} > 0$. Then, we formulate our problem as the following convex program.

$$\min \sum_{O_{i,j} \in \mathcal{O}} w_{i,j} s_{i,j}^{\alpha-1} \tag{5.1}$$

$$\sum_{O_{i,j} \in P_i} \frac{w_{i,j}}{s_{i,j}} \leq d \qquad P_i \in \mathcal{P} \tag{5.2}$$

$$\sum_{O_{i,j} \in J_j} \frac{w_{i,j}}{s_{i,j}} \leq d \qquad J_j \in \mathcal{J} \tag{5.3}$$

$$s_{i,j} \geq 0 \qquad O_{i,j} \in \mathcal{O} \tag{5.4}$$

The term (5.1) is the total energy consumption of all the operations which is our objective function. The constraints (5.2) and (5.3) enforce that all the operations of each processor and each job, respectively, are executed during the interval $[0, d)$. Due to Lemma 5.1, these constraints ensure that the optimal solution of the convex program is a feasible optimal solution for the problem $\mathbf{S}, \mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|\mathbf{E}$. The constraints (5.4) forbid negative speeds. The objective function is convex for $\alpha > 2$ while all the constraints are linear. Hence, the above mathematical program is indeed convex.

Note that we can solve the above convex program in polynomial time by applying the Ellipsoid algorithm. Therefore, we propose the Algorithm 5.1 for the problem $\mathbf{S}, \mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|\mathbf{E}$.

Algorithm 5.1

1: Solve the convex program and determine a speed $s_{i,j}$ for each operation $O_{i,j} \in \mathcal{O}$.
2: Set $p_{i,j} = \frac{w_{i,j}}{s_{i,j}}$ for all $O_{i,j} \in \mathcal{O}$.
3: Apply an optimal algorithm for $\mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|-$ with respect to the $p_{i,j}$'s.

In the following, we propose a faster algorithm for solving the convex program instead of using the Ellipsoid algorithm. Our algorithm is based on a primal-dual schema.

In order to describe our algorithm, we apply the KKT conditions to the convex program (see their general form in Appendix A). First, we associate the dual variables λ_i and μ_j, to the constraints (5.2) and (5.3), respectively. Clearly, for every operation $O_{i,j} \in \mathcal{O}$, it must be the case that $s_{i,j} > 0$. Thus, because of the complementary slackness conditions, the dual variables associated to the constraints (5.4) must be equal to zero.

By the stationarity conditions we get that

$$\nabla \left(\sum_{O_{i,j} \in \mathcal{O}} w_{i,j} s_{i,j}^{a-1} \right) + \sum_{P_i \in \mathcal{P}} \lambda_i \cdot \nabla \left(\sum_{O_{i,j} \in P_i} \frac{w_{i,j}}{s_{i,j}} - d \right) + \sum_{J_j \in \mathcal{J}} \mu_j \cdot \nabla \left(\sum_{O_{i,j} \in J_j} \frac{w_{i,j}}{s_{i,j}} - d \right) = 0 \Leftrightarrow$$

$$\sum_{O_{i,j} \in \mathcal{O}} \left((a-1) w_{i,j} s_{i,j}^{a-2} - (\lambda_i + \mu_j) \frac{w_{i,j}}{s_{i,j}^2} \right) \nabla s_{i,j} = 0 \Leftrightarrow$$

$$s_{i,j}^{\alpha} = \frac{\lambda_i + \mu_j}{\alpha - 1} \qquad O_{ij} \in \mathcal{O} \tag{5.5}$$

The complementary slackness conditions for the constraints (5.2) and (5.3) are expressed as follows.

$$\lambda_i \cdot \left(\sum_{O_{i,j} \in P_i} \frac{w_{i,j}}{s_{i,j}} - d \right) = 0 \qquad P_i \in \mathcal{P} \tag{5.6}$$

$$\mu_j \cdot \left(\sum_{O_{i,j} \in J_j} \frac{w_{i,j}}{s_{i,j}} - d \right) = 0 \qquad J_j \in \mathcal{J} \tag{5.7}$$

The KKT conditions give strong relations between the primal and the dual variables. Indeed, the Equations (5.5) connect directly the primal variables $s_{i,j}$ with the dual variables λ_i and μ_j. Intuitively, each dual variable λ_i, $P_i \in \mathcal{P}$, can be considered as the contribution of the processor P_i to the speed of the operations $O_{i,j}$, $J_j \in \mathcal{J}$. In a similar way, each dual variable μ_j, $J_j \in \mathcal{J}$, can be considered as the contribution of the job J_j to the speed of the operations $O_{i,j}$, $P_i \in \mathcal{P}$.

Next, we describe our primal-dual algorithm for solving the convex program. The main idea of the algorithm is to determine the optimal values of dual variables λ_i and μ_j, and hence the speeds of operations, by modifying them greedily. Our algorithm initializes the dual variables according to the following lemma that provides upper and lower bounds for them.

Lemma 5.2. *In any optimal solution of the convex program, it holds that*

$$(i) \qquad 0 \leq \lambda_i \leq (\alpha - 1) \left(\frac{\sum_{O_{i,j} \in P_i} w_{i,j}}{d} \right)^{\alpha} \qquad P_i \in \mathcal{P}$$

$$(ii) \qquad 0 \leq \mu_j \leq (\alpha - 1) \left(\frac{\sum_{O_{i,j} \in J_j} w_{i,j}}{d} \right)^{\alpha} \qquad J_j \in \mathcal{J}$$

Proof. The lower bound on every λ_i and μ_j comes from the fact that any optimal solution of the convex program satisfies the KKT conditions and, as a result, the dual feasibility conditions.

Consider a processor $P_i \in \mathcal{P}$. As we search for an upper bound on λ_i, we assume that $\lambda_i > 0$. Hence, by (5.6) and (5.5), we have, respectively, that

$$\sum_{O_{i,j} \in P_i} \frac{w_{i,j}}{s_{i,j}} - d = 0 \Leftrightarrow \sum_{O_{i,j} \in P_i} \frac{w_{i,j}}{\sqrt[\alpha]{\frac{\lambda_i + \mu_j}{\alpha - 1}}} = d$$

To obtain the upper bound on λ_i, we can consider that the speed of each operation $O_{i,j} \in P_i$ depends only on the contribution of the processor P_i, that is $\mu_j = 0$ for all $O_{i,j} \in P_i$. Hence, we have that

$$\sum_{O_{i,j} \in P_i} \frac{w_{i,j}}{\sqrt[\alpha]{\frac{\lambda_i}{\alpha - 1}}} \geq d \Leftrightarrow \lambda_i \leq (\alpha - 1) \left(\frac{\sum_{O_{i,j} \in P_i} w_{i,j}}{d} \right)^{\alpha}$$

We can upper bound every γ_j with similar arguments. □

Based on the previous lemma, we initialize each dual variable λ_i, $P_i \in \mathcal{P}$, to its lower bound and each dual variable μ_j, $J_j \in \mathcal{J}$, to its upper bound. Given these initial values, the obtained schedule may not be feasible. More specifically, the total processing time of all the operations of some processor P_i may be more than d, i.e. $\sum_{O_{i,j} \in P_i} \frac{w_{i,j}}{\sqrt[\alpha]{\frac{\mu_j}{\alpha-1}}} > d$ and there might be more than one such processors. For such a processor P_i, we increase λ_i so that that the total processing time of P_i's operations becomes exactly d, i.e. $\sum_{O_{i,j} \in P_i} \frac{w_{i,j}}{\sqrt[\alpha]{\frac{\lambda_i+\mu_j}{\alpha-1}}} = d$. We refer to this step as an *infeasible-to-feasible* step. The increasing of λ_i might have as a result some jobs to become non-tight, i.e. $\sum_{O_{i,j} \in J_j} \frac{w_{i,j}}{\sqrt[\alpha]{\frac{\lambda_i+\mu_j}{\alpha-1}}} < d$. If there is a non-tight job J_j whose dual variable μ_j is positive, then the corresponding equation 5.7 of the complementary slackness conditions is not satisfied. For such a job J_j, we decrease μ_j until it becomes equal to the maximum between zero and the value of μ_j needed so that J_j becomes again tight, i.e. $\sum_{O_{i,j} \in J_j} \frac{w_{i,j}}{\sqrt[\alpha]{\frac{\lambda_i+\mu_j}{\alpha-1}}} = d$. We refer to this step as a *non-tight-to-tight* step. The decreasing of μ_j's has as a result some processors to become non-feasible, and we go on with an infeasible-to-feasible step and so on. The criterion to terminate this procedure is when after a non-tight-to-tight step all the complementary slackness conditions are satisfied. A formal description of the above procedure is given in Algorithm 5.2.

Algorithm 5.2

1: For each $P_i \in \mathcal{P}$, set $\lambda_i = 0$.
2: For each $J_j \in \mathcal{J}$, set $\mu_j = (\alpha - 1)\left(\frac{\sum_{O_{i,j} \in J_j} w_{i,j}}{d}\right)^{\alpha}$.
3: **while** the complementary slackness conditions are not satisfied **do**
4: **for** each $P_i \in \mathcal{P}$ such that the processor P_i is not feasible **do**
5: Choose λ_i such that $\left(\sum_{O_{i,j} \in J_j} \frac{w_{i,j}}{\sqrt[\alpha]{\frac{\lambda_i+\mu_j}{\alpha-1}}} - d\right) = 0$;
6: **for** each $J_j \in \mathcal{J}$ such that the job J_j is not tight **do**
7: Choose the maximum value of μ_j such that $\mu_j \cdot \left(d - \sum_{O_{i,j} \in J_j} \frac{w_{i,j}}{\sqrt[\alpha]{\frac{\lambda_i+\mu_j}{\alpha-1}}}\right) = 0$;

Note that, the algorithm modifies a dual variable λ_i only if the processor P_i is non-feasible in such a way to make it feasible (and tight). In order to accomplish this, the speed of each operation $O_{i,j} \in P_i$ is increased by increasing λ_i. By the definition of the algorithm, P_i can be in a feasible and non-tight state only if $\lambda_i = 0$. In a similar way the algorithm modifies a dual variable μ_j only if the job J_j is non-tight (and feasible) in such a way to make it tight. To do this, the speed of each operation $O_{i,j} \in J_j$ is decreased by decreasing μ_j. By the definition of the algorithm, J_j cannot be in an infeasible state. Based on these observations, the following lemma follows.

Lemma 5.3.
(i) For each $P_i \in \mathcal{P}$, the value of λ_i is always non-decreasing.
(ii) For each $J_j \in \mathcal{J}$, the value of μ_j is always non-increasing.

Theorem 5.1. *Algorithm 5.2 converges to an optimal solution of the convex program.*

Proof. In each iteration the algorithm modifies at least one dual variable; otherwise the complementary slackness conditions are satisfied and the algorithm terminates. By Lemma 5.3 the modification of the dual variables is monotone, while by Lemma 5.2 there are well-defined lower and upper bounds for them. Therefore, the algorithm terminates.

In order to show that the algorithm converges in an optimal solution, we just have to observe that the final solution satisfies the KKT conditions. The stationarity conditions (5.5) hold for any operation $O_{i,j}$ as its speed is set as $s_{i,j} = \sqrt[\alpha]{\frac{\lambda_i+\mu_j}{\alpha-1}}$. The complementary slackness conditions (5.6) hold since after the final non-tight-to-tight step any processor P_i is either tight or $\lambda_i = 0$; if not then the algorithm would have executed a new iteration. The complementary slackness conditions (5.7) hold since we force that $\mu_j \cdot \left(d - \sum_{O_{i,j} \in J_j} \frac{w_{i,j}}{\sqrt[\alpha]{\frac{\lambda_i+\mu_j}{\alpha-1}}} \right) = 0$ in the last non-tight-to-tight step. □

5.1.2 Experimental Evaluation of the Primal-Dual Algorithm

In the following we test our primal-dual algorithm experimentally towards two directions. The first direction is to observe the behavior of our algorithm when the size of the instance increases. The second direction is to compare the execution time of the primal-dual approach against the execution time of a baseline algorithm which is a commercial solver that solves directly the corresponding convex program.

System Specification and Benchmark Generation

Our simulations have been performed on a machine with a CPU Intel Xeon X5650 with 8 cores, running at 2.67GHz. The operating system of the machine is a Linux Debian 6.0. We used Matlab with the CVX toolbox. The solver used for the convex program is SeDuMi. For both our algorithm and the convex program, we set $\varepsilon = 10^{-7}$ to be the desired accuracy of the returned solution.

The instance of the problem consists of a matrix $m \times n$ that corresponds to the work of the operations, the value of α and the deadline d. However, we experiment with two more parameters: (i) the density p of the instance, that is the number of non-zero work operations, and (ii) the range $[1, w_{\max}]$ of the values of works.

We have considered several combinations for the parameters m, $1 \leq m \leq 50$, and n, $1 \leq n \leq 200$. For each combination, we have first decided randomly with probability p if there is a non-zero work operation in each position of the $m \times n$ matrix. The value of p has been selected to be 0.5 or 0.75 or 1. If the created instance did not correspond to the selected values of m and n, we rejected it and we replaced it by another. In other words, we reject a matrix iff there exists a line or a column in which each value is equal to zero. Then, for each operation with non-zero work, we selected at random an integer in the range of $[1, w_{\max}]$. Note here that $w_{\max}$ and the deadline d are strongly related. Indeed, given a matrix of works and a deadline d, if we increase all works and the deadline by the same factor, then the optimal solutions of the two instances will tend to have very similar (if not the same) speeds and energy consumption. For this reason, we have fixed

the value of $d = 1000$ and we examined three different values for $w_{\max}$, i.e., $w_{\max} = 10$, $w_{\max} = 50$ and $w_{\max} = 100$. These values are selected, in general, in the direction of creating instances in which the average speed in the optimal solution is greater than one, almost equal to one and smaller than one, respectively. Finally, as in most applications the value of α is between two and three, we used three different values for it, that is $\alpha = 2$, $\alpha = 2.5$ and $\alpha = 3$.

For each combination of parameters we have repeated the experiments with 30 different matrices. All results we present below, concern the average of these 30 instances.

Results

The main goal of our experiments is to study the behavior of the primal-dual algorithm when the size of the instance increases. However, during our experiments we noticed that the speed of convergence strongly depends on the relation between the number of jobs n and the number of processors m.

In Table 5.1, we show how the size of the instance affects the number of modifications of the dual variables made by the primal-dual algorithm. We observe that, if $n > m$ then the number of modifications increases linearly with the size of the instance (see also Figure 5.1 for a graphical representation). Moreover, the parameters α, $w_{\max}$ and the density p do not play any role to the number of modifications.

n	$m = 5$	$m = 10$	$m = 15$	$m = 20$	$m = 25$	$m = 30$	$m = 40$	$m = 50$
5	40101	1	2	2	2	2	2	2
10	151	279611	3	4	3	4	4	4
20	255	295	384	–	34	7	7	10
30	355	410	443	500	593	–	12	15
40	455	510	565	572	640	756	–	32
50	555	610	665	720	768	755	947	–
60	655	710	765	820	872	864	1040	1294
70	755	810	865	920	975	1030	1034	1250
100	1055	1110	1165	1220	1275	1330	1440	1495
150	1555	1610	1665	1720	1775	1830	1940	2050
200	2055	2110	2165	2220	2275	2330	2440	2550

Table 5.1: The number of modifications of the dual variables done by the primal-dual algorithm. The values of the table correspond to $\alpha = 2$, $w_{\max} = 10$, $p = 1$. Each entry of the table is the average over 30 instances. The empty entries correspond to cases with $m = n$ and take time longer than 30 minutes each and are interrupted.

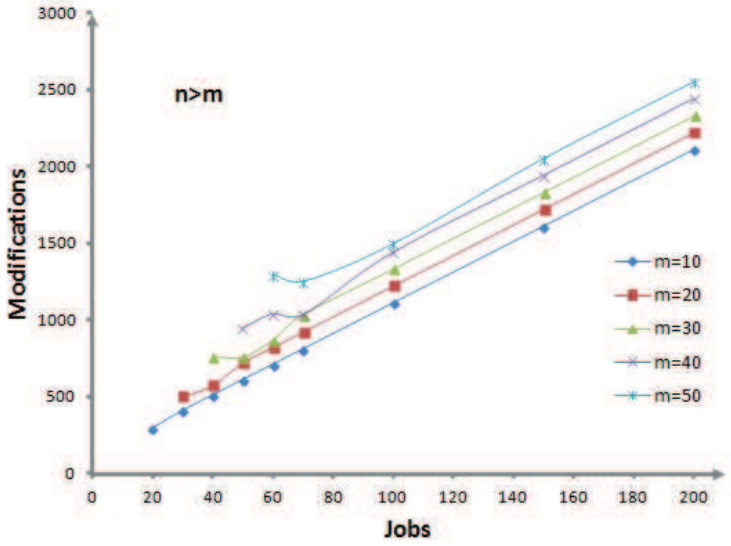


Figure 5.1: The number of modifications of the dual variables made by the primal-dual algorithm if $n > m$ ($\alpha = 2$, $w_{\max} = 10$, $p = 1$).

Note also that if $n < m$ then the number of modifications increases linearly with the size of the instance. In fact the two cases $n > m$ and $n < m$ should be symmetric. However, the initialization step of our algorithm breaks this symmetry. Recall that the algorithm initializes the dual variables that correspond to processors (λ_i's) to zero and the dual variables that correspond to jobs (μ_j's) to their upper bounds. In the case where $n < m$, we expect to have all jobs tight and most of the processors non-tight in the optimal schedule of a random instance. Hence, the number of non-zero λ_i's is expected to be very small. The initialization step helps in this direction, and this is the reason why the number of modifications is very small if $n < m$.

However, if $n = m$ the behavior of our algorithm completely changes. For example, for $m = 10$ and $n = 10$ we need 279611 modifications, while for $m = 10$ and $n = 20$ we need only 295. Even more, for $m = n = 20$ the primal-dual algorithm does not even converges in 30 minutes. Furthermore, if $m = n$ then the parameters α, $w_{\max}$ and p affect the convergence of the algorithm. For example, in the case where $m = 10$ and $n = 10$, then Table 5.2 shows the number of modifications of the dual variables performed by our algorithm when we fix the two of the three parameters. Note that in the last line of the table, the algorithm did not terminate within the time threshold.

Parameters		Modifications
$\alpha = 2$ $w_{\max} = 10$	$p = 0.5$ $p = 0.75$ $p = 1$	344 23915 179611
$w_{\max} = 10$ $p = 1$	$\alpha = 2$ $\alpha = 2.5$ $\alpha = 3$	279611 59785 10716
$\alpha = 2$ $p = 1$	$w_{\max} = 10$ $w_{\max} = 50$ $w_{\max} = 100$	279611 406608 –

Table 5.2: The table shows how the parameters α, p and w_{max} affect the performance of the primal-dual algorithm when $n = m = 10$.

In Table 5.3 we give a comparison of the execution time of the primal-dual algorithm with the execution time of solving directly the convex program using the SeDuMi solver in Matlab. We observe again the difference between $n \neq m$ and $n = m$. In the first case, our algorithm highly outperforms the solver (see Figure 5.2). In the second case, our algorithm does not even terminates within 30 minutes if $n = m = 20$, while the execution time of the solver is not affected by this. Note also that the execution time of the solver as well as the execution time of the primal-dual algorithm when $n = m$ depend on the parameters α, $w_{\max}$ and p.

n	$m = 10$		$m = 20$		$m = 30$		$m = 40$		$m = 50$	
	CP	PD	CP	PD	CP	PD	CP	PD	CP	PD
5	0.59	0.00	0.99	0.00	1.41	0.01	1.83	0.01	2.11	0.01
10	1.22	147.93	1.26	0.01	1.81	0.01	2.42	0.01	2.59	0.01
20	1.25	0.06	3.12	–	2.57	0.02	3.11	0.02	3.92	0.03
30	1.72	0.08	2.58	0.12	5.57	–	4.36	0.03	5.30	0.04
40	2.17	0.10	3.28	0.13	4.38	0.21	8.31	–	6.48	0.05
50	2.67	0.12	4.00	0.16	5.19	0.19	6.72	0.33	11.49	–
60	3.47	0.15	4.96	0.18	6.72	0.23	8.39	0.29	9.87	0.47
70	3.86	0.16	5.99	0.21	7.73	0.26	9.84	0.28	11.42	0.40
100	5.85	0.22	8.62	0.27	11.85	0.32	13.86	0.38	17.56	0.42
150	9.31	0.31	14.34	0.38	19.30	0.47	24.66	0.52	31.10	0.56
200	12.89	0.42	19.87	0.51	28.78	0.59	36.83	0.68	46.31	0.74

Table 5.3: A comparison of the execution time of the primal-dual approach (PD) with the execution time of the SeDuMi solver for convex programs (CP). The execution times are computed in *seconds*. The values of the table correspond to $\alpha = 2$, $w_{\max} = 10$, $p = 1$. Each entry of the table is the average over 30 instances. The empty entries correspond to cases with $m = n$ and take time longer than 30 minutes each and are interrupted.

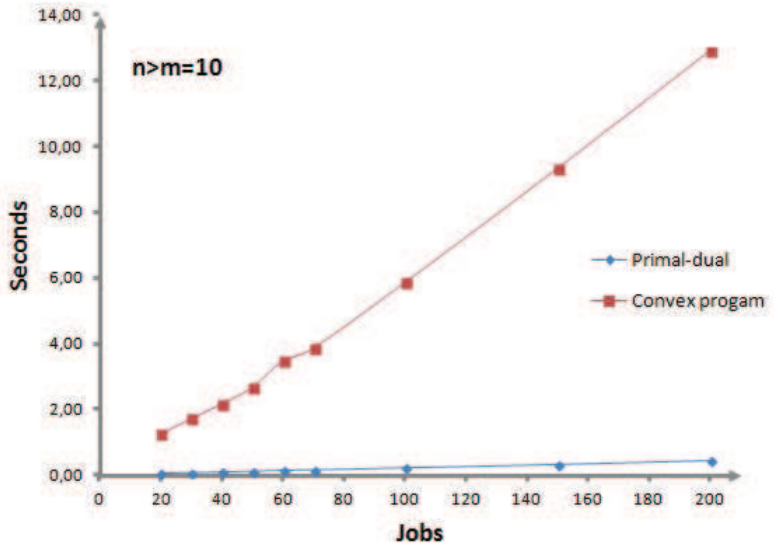


Figure 5.2: A comparison of the execution times of the primal-dual algorithm and the SeDuMi solver for convex programs if $n > m$ ($m = 10$, $\alpha = 2$, $w_{\max} = 10$, $p = 1$).

The results presented above motivated us to further explore the case $n = m$. For this reason, we performed more experiments for $m = 10, 20, 30$ and $n = m-5, m-4, \ldots, m+4, m+5$. The results of these experiments are shown in Figure 5.3. The horizontal axis corresponds to the difference $m - n$, while the vertical axis corresponds to the logarithm of the modifications of the dual variables made by our algorithm.

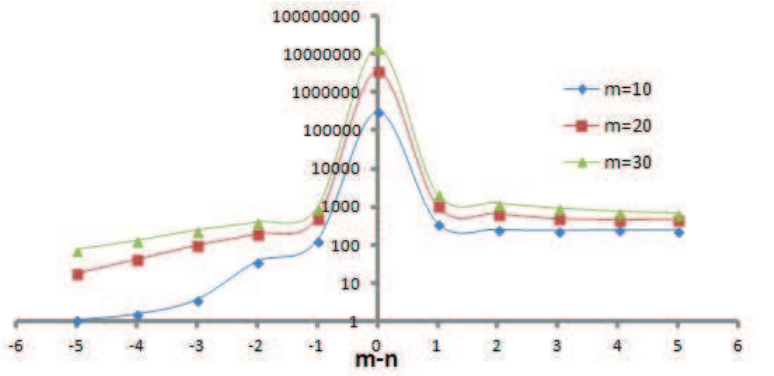


Figure 5.3: The vertical axis represents the logarithm of the modifications of the dual variables made by the primal-dual algorithm ($\alpha = 2$, $w_{\max} = 10$, $p = 1$).

We observe that the behavior of the primal-dual algorithm dramatically changes when $n = m$, while there is a much smaller perturbation when $n = m \pm 1$ and $n = m \pm 2$. In all other cases the number of modifications seems to increase linearly with the size of the instance. The problem with the case where $n = m$ probably occurs because in an optimal solution of a random instance almost all processors and jobs are tight, that is the total execution time of each processor and each job is equal to the deadline d. In other

words, all λ_i's and μ_j's are expected to be non-zero. The primal-dual algorithm, in each iteration "corrects" first the values of λ_i's and then the values of μ_j's. As all of them are expected to be non-zero in the optimal solution the required precision plays a significant role to the speed of the convergence of the algorithm.

5.1.3 Optimal Algorithm based on Minimum Convex Cost Flow

Next, we describe an optimal algorithm for $\mathbf{S}, \mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|\mathbf{E}$ which comes from a formulation of the problem as a convex cost flow problem. The main difficulty in establishing this formulation is the fact that we are not able tos compute directly the total processing time T^* of all the operations in an optimal schedule. We overcome this difficulty by calculating the value of T^* algorithmically. Specifically, we apply a sort of binary search with repeated convex cost flow computations. Once we have calculated T^*, a convex cost flow calculation gives the speeds, and hence the execution times, of the operations in an optimal schedule for $\mathbf{S}, \mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|\mathbf{E}$. In order to construct a feasible schedule, we then apply an optimal algorithm for the feasibility problem $\mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|-$.

Recall that, in $\mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|-$, we are given a set of n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ and a set of m parallel processors $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$. Each job $J_j \in \mathcal{J}$ is composed of m operations $O_{1,j}, O_{2,j}, \ldots, O_{m,j}$. The operation $O_{i,j}$ of the job $J_j \in \mathcal{J}$ has an amount of work $w_{i,j} \geq 0$ and it must be entirely executed on the processor $P_i \in \mathcal{P}$. Two operations of the same job must not be executed simultaneously. Preemptions of jobs are allowed. The objective of the problem is to construct a feasible schedule or decide that such a schedule does not exist.

Next, we formulate $\mathbf{S}, \mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|\mathbf{E}$ as a convex cost flow problem (the definition of the latter problem can be found in the Appendix C). We construct a graph $G = (V, A)$ which consists of a source node s, a destination node t, a job node J_j, for each job $J_j \in \mathcal{J}$, and a processor node P_i, for each processor $P_i \in \mathcal{P}$. The graph G contains an arc (s, J_j) with capacity d for each $J_j \in \mathcal{J}$, an arc (P_i, t) with capacity capacity d for each processor $P_i \in \mathcal{P}$ and an arc (J_j, P_i) of infinite capacity, for every $P_i \in \mathcal{P}$ and $J_j \in \mathcal{J}$ such that $w_{i,j} > 0$. Apart from a capacity, each arc $e \in A$ of G comes with a convex cost function $g_e(x)$ which specifies the cost incurred if x units of flow cross the arc e. The cost functions of the arcs are defined as follows.

- $g_{(s,J_j)}(x) = 0$, for all $J_j \in \mathcal{J}$,
- $g_{(P_i,t)}(x) = 0$, for all $P_i \in \mathcal{P}$, and
- $g_{(J_j,P_i)}(x) = \frac{w_{i,j}^{\alpha}}{x^{\alpha-1}}$, for all $J_j \in \mathcal{J}$ and $P_i \in \mathcal{P}$ such that $w_{i,j} > 0$.

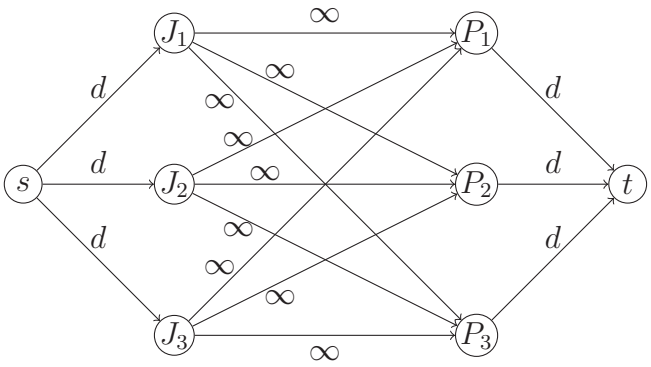


Figure 5.4: The graph for the convex cost flow formulation of $\mathbf{S}, \mathbf{0}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|\mathbf{E}$. The arcs (s, J_j) and (P_i, t) have zero cost functions while each arc (J_j, P_i) has cost function $g_{(J_j,P_i)} = \frac{w_{i,j}^{\alpha}}{x^{\alpha-1}}$.

We can imagine that any feasible (s,t)-flow in the graph G corresponds to a feasible schedule for $\mathbf{S}, \mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|\mathbf{E}$. The flow traversing the arcs of G corresponds to the execution time of the jobs. Consider a feasible (s,t)-flow $\mathcal{F}$ in G and let f_e be the amount of flow that crosses the arc $e \in A$ according to $\mathcal{F}$. The value $f_{(J_j,P_i)}$ corresponds to the execution time of the operation $O_{i,j}$, i.e. $\frac{w_{i,j}}{f_{(J_j,P_i)}}$ is the speed of $O_{i,j}$ and $\frac{w_j^{\alpha}}{f_{(J_j,P_i)}^{\alpha-1}}$ is the energy consumed for the execution of $O_{i,j}$. Furthermore, $f_{(s,J_j)}$ represents the total execution time of all the operations of the job J_j. Similarly, $f_{(P_i,t)}$ corresponds to the total time that P_i executes any operation. Hence, the total flow that leaves the source node s and arrives to the destination node t corresponds to the total execution time of all the operations.

In order to complete our convex cost flow formulation for $\mathbf{S}, \mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|\mathbf{E}$, it remains to specify the amount of flow T^* that has to be sent from s to t. Notice that T^* must be the total processing time of all the operations in an optimal schedule for $\mathbf{S}, \mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|\mathbf{E}$. Unfortunately, the value T^* cannot be computed through a straightforward formula. However we describe how to compute it algorithmically at the end of this section. For the moment, we assume that we can indeed compute T^* efficiently. Then, we propose the following algorithm for $\mathbf{S}, \mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|\mathbf{E}$.

Algorithm 5.3

1: Construct the graph G.
2: Compute the total execution time of all operations T^* in an optimal schedule.
3: Find a convex cost (s,t)-flow $\mathcal{F}$ of value T^* in G.
4: Determine the processing time $p_{i,j}$ of each operation.
5: Apply an algorithm for $\mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|-$ to find a feasible schedule with respect to the $p_{i,j}$'s.

Let us, now, show that our algorithm is optimal.

Theorem 5.2. *Algorithm 5.3 finds an optimal schedule for* $\mathbf{S}, \mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|\mathbf{E}$.

Proof. We first prove that there exists a feasible schedule of total execution time T for $\mathbf{S}, \mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|\mathbf{E}$ if and only if there is a feasible (s,t)-flow $\mathcal{F}$ of value T in the graph G, for any $T > 0$.

Suppose that there exists a feasible schedule of total execution time T. Let $p_{i,j}$ be the execution time of the operation $O_{i,j}$. Hence, the speed of $O_{i,j}$ is $\frac{w_{i,j}}{p_{i,j}}$. Then, consider the flow in G which is defined as follows.

- $f_{(s,J_j)} = \sum_{P_i \in \mathcal{P}} p_{i,j}$ for all $J_j \in \mathcal{J}$
- $f_{(J_j,P_i)} = p_{i,j}$ for all $P_i \in \mathcal{P}$ and $J_j \in \mathcal{J}$
- $f_{(P_i,t)} = \sum_{J_j \in \mathcal{J}} p_{i,j}$ for all $P_i \in \mathcal{P}$

Recall that every operation must be executed during the interval $[0, d)$. Because of the open shop constraint, it must hold that $\sum_{P_i \in \mathcal{P}} p_{i,j} \leq d$. Moreover, due to the fact that each processor can execute at most one operation at each time, we have that $\sum_{J_j \in \mathcal{J}} p_{i,j} \leq d$. Therefore, the above flow is of value T and it is a feasible flow in the graph G.

To the other direction, assume that there exists a feasible flow of value T in G. We can then define a feasible schedule for $\mathbf{S}, \mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|\mathbf{E}$ by setting the processing time of each operation $O_{i,j}$ to be equal to $f_{(J_j,I_i)}$, i.e. by setting the speed of $O_{i,j}$ equal to $\frac{w_{i,j}}{f_{(J_j,I_i)}}$. Since the flow is feasible, it must hold that $\sum_{P_i \in \mathcal{P}} f_{(J_j,I_i)} \leq d$ and $\sum_{J_j \in \mathcal{J}} f_{(J_j,I_i)} \leq d$. By Lemma 5.1, we can construct a feasible schedule for $\mathbf{S}, \mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|\mathbf{E}$.

We conclude the proof with the optimality of our algorithm. Among the feasible flows of value T^*, the algorithm finds the one which minimizes the term $\sum_{P_i \in \mathcal{P}} \sum_{J_j \in \mathcal{J}} \frac{w_{i,j}^{\alpha}}{f_{(J_j,P_i)}^{\alpha-1}}$. In other words, given our convex cost flow formulation, the algorithm finds the schedule with the minimum energy among the schedules of total execution time T^*. But, we have assumed that there exists an optimal schedule of total execution time equal to T^* and we can compute T^* efficiently. Hence, our algorithm is optimal. □

It remains to show how we can compute the total execution time T^* of all operations in an optimal schedule for $\mathbf{S}, \mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|\mathbf{E}$. Let us first introduce some additional notation. Given a feasible schedule $\mathcal{S}$, we denote by $p_{i,j}$ the execution time of the operation $O_{i,j}$, for $P_i \in \mathcal{P}$ and $J_j \in \mathcal{J}$, in $\mathcal{S}$. Then, let $\vec{p} = (p_{1,1}, p_{2,1}, \ldots, p_{m,1}, p_{1,2}, \ldots, p_{m,n})$ be the vector of the execution times of all the operations in $\mathcal{S}$. Then, let $T(\vec{p}) = \sum_{P_i \in \mathcal{P}} \sum_{J_j \in \mathcal{J}} p_{i,j}$ and $E(\vec{p}) = \sum_{P_i \in \mathcal{P}} \sum_{J_j \in \mathcal{J}} \frac{w_{i,j}^{\alpha}}{p_{i,j}^{\alpha-1}}$ be the functions that map any vector of execution times $\vec{p}$ to the total execution time and the total energy consumption of the schedule $\mathcal{S}$. Note that, $E(\vec{p})$ is convex with respect to the vector $\vec{p}$ as a sum of convex functions. Furthermore, we define the function $E^*(T) = \min\{E(\vec{p}) : T(\vec{p}) = T\}$ which indicates the minimum energy consumption when the sum of execution times of all operations must be equal to T.

Lemma 5.4. *$E^*(T)$ is convex with respect to T.*

Proof. Consider three values of total execution times $T_1, T_2, T_3 > 0$, and let $\vec{p_1}$, $\vec{p_2}$, $\vec{p_3}$ be three corresponding optimal vectors of execution times, respectively. That is, $T(\vec{p_1}) = T_1$, $T(\vec{p_2}) = T_2$, $T(\vec{p_3}) = T_3$ and $E^*(T_1) = E(\vec{p_1})$, $E^*(T_2) = E(\vec{p_2})$, $E^*(T_3) = E(\vec{p_3})$. In other words, $\vec{p_1}$, $\vec{p_2}$, $\vec{p_3}$ define optimal schedules (w.r.t. to minimizing the total energy consumption) given that the total execution time of all the operations must be equal to T_1, T_2, T_3, respectively. Without loss of generality, assume that $T_1 \leq T_3 \leq T_2$. Clearly, there must be a $\theta \in [0, 1]$ such that $T_3 = \theta T_1 + (1 - \theta)T_2$. Consider, now, the vector $\vec{p} = \theta\vec{p_1} + (1 - \theta)\vec{p_2}$. As $T(\vec{p})$ is linear with respect to $\vec{p}$, it holds that

$$T(\vec{p}) = T(\theta\vec{p_1} + (1 - \theta)\vec{p_2}) = \theta T(\vec{p_1}) + (1 - \theta)T(\vec{p_2}) = T(\vec{p_3}) = T_3$$

By the definition of $E^*(T)$, it holds that $E^*(T_3) \leq E(\vec{p})$. Moreover, recall that the function $E(\vec{p})$ is convex with respect to $\vec{p}$. In all, we have that

$$\begin{aligned} E^*(\theta T_1 + (1 - \theta)T_2) &= E^*(T_3) \\ &\leq E(\vec{p}) \\ &= E\left(\theta\vec{p_1} + (1 - \theta)\vec{p_2}\right) \\ &\leq \theta E(\vec{p_1}) + (1 - \theta)E(\vec{p_2}) \\ &= \theta E^*(T_1) + (1 - \theta)E^*(T_2) \end{aligned}$$

and, hence, the function $E^*(T)$ is convex with respect to T. □

Next, we give the search algorithm that finds the value $T^* = \arg\min_T\{E^*(T)\}$. Consider any $T_1, T_2, T_3 > 0$ such that $T_1 < T_3$ and $T_2 = \frac{T_1+T_3}{2}$. Clearly, $T_1 < T_2 < T_3$. As $E^*(T)$ is convex with respect to T, we have that $E^*(T_2) \leq \frac{E^*(T_1)+E^*(T_3)}{2}$. Therefore, it follows that either $E^*(T_2) \leq E^*(T_1)$ or $E^*(T_2) \leq E^*(T_3)$ (or both). If only the first is true, then we reduce our search space to $[T_2, T_3]$. Accordingly, if only the second is true, then we reduce our search space to $[T_1, T_2]$. Finally, if both are true, then we reduce our search space to one of the following intervals: $[T_1, T_2]$, $[T_2, T_3]$ or $[\frac{T_1+T_2}{2}, \frac{T_2+T_3}{2}]$. Notice that $\frac{T_1+T_2}{2} < T_2 < \frac{T_2+T_3}{2}$ and that $(\frac{T_1+T_2}{2} + \frac{T_2+T_3}{2})/2 = T_2$. Thus, due to the convexity of $E^*(T)$, we have that $E^*(T_2) \leq (E^*(\frac{T_1+T_2}{2}) + E^*(\frac{T_2+T_3}{2}))/2$. So, it must be the case that either $E^*(T_2) \leq E^*(\frac{T_1+T_2}{2})$ or $E^*(T_2) \leq E^*(\frac{T_2+T_3}{2}))$ (or both). If only the first is true, then the search space is reduced to $[T_2, T_3]$. Similarly, if only the second is true, then the search space is reduced to $[T_1, T_2]$. Finally, if both are true, then the search space is reduced to $[\frac{T_1+T_2}{2}, \frac{T_2+T_3}{2}]$. The correctness of all the cases is based on the fact that $E^*(T)$ is convex. The Algorithm 5.4 performs this procedure. All the possible cases of the search space of the binary search are illustrated in Figure 5.5. The values T_1, T_2 and T_3 are initialized as follows: $T_1 = 0$, $T_2 = \frac{T_{UB}}{2}$ and $T_3 = T_{UB}$, respectively, where T_{UB} is an upper bound on the sum of execution times for all operations in any optimal schedule. For instance, $T_{UB} = m \cdot d$.

Algorithm 5.4

$T_{UB} = m \cdot d$.
$T_1 = 0$, $T_2 = \frac{T_{UB}}{2}$, $T_3 = T_{UB}$.
while $T_3 - T_1 > \epsilon$ **do**
  Compute the values $E^*(T_1)$, $E^*(T_2)$ and $E^*(T_3)$ by computing convex cost flows of values T_1, T_2 and T_3, respectively, in the graph G.
  if $E^*(T_1) \geq E^*(T_2)$ and $E^*(T_2) > E^*(T_3)$ **then**
    $T_1' = T_2$, $T_2' = \frac{T_2+T_3}{2}$, $T_3' = T_3$.
  if $E^*(T_1) < E^*(T_2)$ and $E^*(T_2) \leq E^*(T_3)$ **then**
    $T_1' = T_1$, $T_2' = \frac{T_1+T_2}{2}$, $T_3' = T_2$.
  if $E^*(T_1) \geq E^*(T_2)$ and $E^*(T_2) \leq E^*(T_3)$ **then**
    Compute the values $E^*(\frac{T_1+T_2}{2})$ and $E^*(\frac{T_2+T_3}{2})$ by computing convex cost flows of values $\frac{T_1+T_2}{2}$ and $\frac{T_2+T_3}{2}$, respectively, in the graph G.
    if $E^*(\frac{T_1+T_2}{2}) \geq E^*(T_2)$ and $E^*(T_2) > E^*(\frac{T_2+T_3}{2})$ **then**
      $T_1' = T_2$, $T_2' = \frac{T_2+T_3}{2}$, $T_3' = T_3$.
    if $E^*(\frac{T_1+T_2}{2}) < E^*(T_2)$ and $E^*(T_2) \leq E^*(\frac{T_2+T_3}{2})$ **then**
      $T_1' = T_1$, $T_2' = \frac{T_1+T_2}{2}$, $T_3' = T_2$.
    if $E^*(\frac{T_1+T_2}{2}) \geq E^*(T_2)$ and $E^*(T_2) \leq E^*(\frac{T_2+T_3}{2})$ **then**
      $T_1' = \frac{T_1+T_2}{2}$, $T_2' = T_2$, $T_3' = \frac{T_2+T_3}{2}$.
  $T_1 = T_1'$, $T_2 = T_2'$, $T_3 = T_3'$.
return T_2;

Lemma 5.5. *Algorithm 5.4 returns a value T^* such that the term $E^*(T^*)$ is minimized.*

Proof. At each iteration of the algorithm, the search space is reduced. This reduction is accomplished by removing one of the intervals $[T_1, T_2)$, $(T_2, T_3]$, or $[T_1, \frac{T_1+T_2}{2}) \cup (\frac{T_2+T_3}{2}, T_3]$ from the search space. In order to establish the correctness of our algorithm, it suffices to show that, at the end of each iteration, there is a value T^* in the algorithm's remaining search space that minimizes the term $E^*(T^*)$.

Consider some iteration of Algorithm 5.4 and suppose that it removes the interval $(T_2, T_3]$ from the search space. We assume for the sake of contradiction that $T^* \in (T_2, T_3]$, for any $T^* = \arg\min_T\{E^*(T)\}$. Since the interval $(T_2, T_3]$ is removed by the algorithm, we have one of the following cases:

- either $E^*(T_1) < E^*(T_2) \leq E^*(T_3)$,
- or $E^*(T_2) \leq E^*(T_1)$, $E^*(T_2) \leq E^*(T_3)$ and $E^*(\frac{T_1+T_2}{2}) < E^*(T_2)$.

Initially, we consider the former case. Let T^* be the total execution time of an optimal solution. Since $T^* \in (T_2, T_3]$, it holds that $T_2 \in [T_1, T^*)$. Therefore, we know that there is a $\theta \in [0, 1]$ such that $T_2 = \theta T_1 + (1-\theta)T^*$. Then, due to the convexity of the function $E^*(T)$, we have that $E^*(T_2) \leq \theta E^*(T_1) + (1-\theta)E^*(T^*)$. But, $E^*(T_1) < E^*(T_2)$ and, as a result, $E^*(T_2) < \theta E^*(T_2) + (1-\theta)E^*(T^*)$, or, equivalently, $E^*(T_2) < E^*(T^*)$. But, this contradicts the fact that T^* minimizes $E^*(T^*)$.

We, now, consider the latter case. Let $T^* \in (T_2, T_3]$ be some value minimizing $E^*(T^*)$. Then, by arguing as before, there is a $\theta \in [0, 1]$ such that $T_2 = \theta(\frac{T_1+T_2}{2}) + (1-\theta)T^*$. By

the convexity of the function $E^*(T)$ and the fact that $E^*(\frac{T_1+T_2}{2}) < E^*(T_2)$, which reach a contradiction on the optimality of T^* as before.

Note that, if the interval ignored by the algorithm is $[T_1, T_2)$, then the fact that there is a value $T^* \in [T_2, T_3]$ minimizing $E^*(T^*)$ can be proved with almost the same manner as in the case where the interval $(T_2, T_3]$ is removed.

Finally, suppose that the algorithm reduces the search space to $[\frac{T_1+T_2}{2}, \frac{T_2+T_3}{2}]$. This happens when $E^*(T_2) \leq E^*(T_1)$, $E^*(T_2) \leq E^*(\frac{T_1+T_2}{2})$, $E^*(T_2) \leq E^*(\frac{T_2+T_3}{2})$ and $E^*(T_2) \leq E^*(T_3)$. Assume for contradiction that $T^* \in (\frac{T_2+T_3}{2}, T_3]$ (note that we must also consider the case where $T^* \in [T_1, \frac{T_1+T_2}{2})$ but it can be handled analogously). Due to the convexity of the function $E^*(T)$, we have that $E^*(\frac{T_2+T_3}{2}) \leq \frac{E^*(T_2)+E^*(T_3)}{2}$. Thus, $E^*(\frac{T_2+T_3}{2}) \leq E^*(T_3)$. Given that $T_2 \leq \frac{T_2+T_3}{2} < T^* \leq T_3$ and $E^*(T_2), E^*(\frac{T_2+T_3}{2}) \leq E^*(T_3)$, we can reach a contradiction on the fact that there is an optimal solution T^* such that $T^* \in (\frac{T_2+T_3}{2}, T_3]$ as before. □

5.2 Energy Minimization in a Job Shop

In this section, we consider the energy minimization problem in a job shop environment in which preemption of operations are allowed, i.e. $\mathbf{S}, \mathbf{J}|\mathbf{r_{i,j}}, \mathbf{d_{i,j}}, \mathbf{pmtn}|\mathbf{E}$, and we present an $(1+\epsilon)\tilde{B}_\alpha$-approximation algorithm. Our algorithm solves the fractional relaxation of an integer configuration linear program (LP) and computes a solution for the problem by applying randomized rounding.

The instance of the problem consists of a set of jobs $\mathcal{J}$, where each job $J_j \in \mathcal{J}$ consists of n_j operations $O_{j,1}, O_{j,2}, \ldots, O_{j,n_j}$, which must be executed in this order. That is, $O_{k+1,j}$ can start only once the operation $O_{j,k}$ has finished. Let $\tilde{n}$ be the number of all the operations, i.e. $\tilde{n} = \sum_{j \in \mathcal{J}} n_j$. Each operation $O_{j,k}$ has an amount of work $w_{j,k}$. Moreover, we are given a set of m heterogeneous processors $\mathcal{P}$. Each operation $O_{j,k}$, $J_j \in \mathcal{J}$ and $1 \leq k \leq n_j$, is associated with a single processor $P_i \in \mathcal{P}$ on which it must be entirely executed. Note that more than one operations of the same job may have to be executed on the same processor. Furthermore, for each operation $O_{j,k}$, we are given a release date $r_{j,k}$ and a deadline $d_{j,k}$. For each $J_j \in \mathcal{J}$, we can assume that $r_{j,1} \leq r_{j,2} \leq \ldots \leq r_{j,n_j}$ as well as $d_{j,1} \leq d_{j,2} \leq \ldots \leq d_{j,n_j}$. Preemptions of operations are allowed. The objective is to find a feasible schedule of minimum energy consumption.

Initially, we formulate the job shop problem as an integer configuration LP. In order to define the notion of a configuration, we discretize the time into a number of slots which is polynomial to the size of the instance and to $1/\varepsilon$. Once we have discretized the time, we consider a variation of our problem in which, during every such slot on a processor $P_i \in \mathcal{P}$, either a single operation is executed or the P_i is idle. The optimal energy consumption of the new problem is at most a factor of $(1+\epsilon)$ the energy consumption of the original job shop problem. Then, we define a configuration as a schedule for a job, i.e. a schedule for all the operations of the job. Due to the convexity of the speed-to-power function of every processor, there is always an optimal schedule such that each operation is executed at a constant speed. Therefore, a well-defined configuration has to specify the set of slots during which every operation of a job is executed.

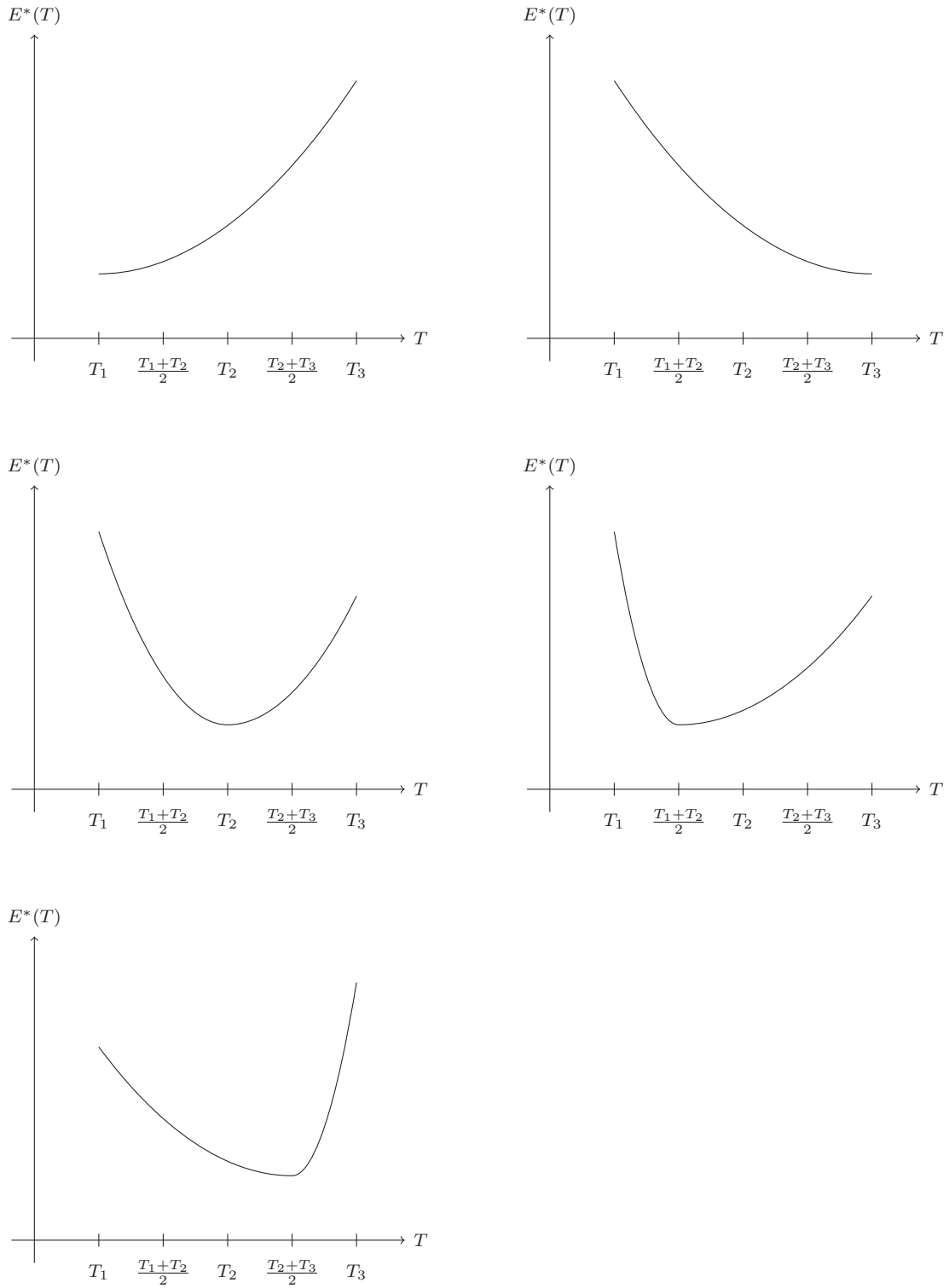


Figure 5.5: The possible cases of the binary search Algorithm 5.4.

We partition the time into intervals as follows. We define the time points $t_0, t_1, \ldots, t_\tau$, in increasing order, where each t_ℓ corresponds to either a release date or a deadline, so that for each release date and deadline of an operation there is a corresponding t_ℓ. Then, we define the intervals $I_\ell = [t_{\ell-1}, t_\ell)$, for $1 \leq \ell \leq \tau$, and we denote by $|I_\ell|$ the length of I_ℓ. Note that there are no release dates or deadlines inside any interval I_ℓ, $1 \leq \ell \leq \tau$. Then we further discretize the time in each interval I_ℓ. In the following lemma we assume that the release dates and the deadlines of all operations are integers.

Lemma 5.6. *There is a feasible schedule with energy consumption at most* $(1+\epsilon)^{\alpha_{max}} \cdot OPT$ *in which each piece of every operation* $O_{j,k}$, $J_j \in \mathcal{J}$ *and* $1 \leq k \leq n_j$, *executed during the interval* I_ℓ, $1 \leq \ell \leq \tau$, *starts and ends at a time point* $t_{\ell-1} + r\frac{\epsilon}{\tilde{n}(1+\epsilon)}|I_\ell|$, *where* $r \geq 0$ *is an integer.*

Proof. Consider an optimal schedule $\mathcal{S}^*$ for the job shop problem. For the interval I_ℓ, $1 \leq \ell \leq \tau$, we define the time points $q_0 < q_1 < q_2 < \ldots < q_u$, where $q_0 = t_{\ell-1}$ and $q_u = t_\ell$, so that each q_p, $0 \leq p \leq u$, corresponds to either a begin time or a completion time of a piece of an operation on any processor during I_ℓ in $\mathcal{S}$, and there is a corresponding time point q_p for every possible begin time and completion time. We call the interval $(q_{p-1}, q_p]$, for $1 \leq p \leq u$, a *slice*. By Baptiste et al. [19], we know that there exists an optimal schedule $\mathcal{S}^*$ with at most $\tilde{n}$ slices during I_ℓ, i.e. $u = O(\tilde{n})$.

We will now transform an optimal schedule $\mathcal{S}^*$ to a feasible schedule $\mathcal{S}$ satisfying the lemma. Consider an interval I_ℓ, $1 \leq \ell \leq \tau$. We first create an idle period of length $\frac{\epsilon}{1+\epsilon}|I_\ell|$. This can be done by increasing the speeds of all processors of all slices in I_ℓ by a factor of $1+\epsilon$. Hence, the energy consumption is at most a factor of $(1+\epsilon)^{\alpha_{max}}$ far from the energy of $\mathcal{S}^*$. In order to obtain $\mathcal{S}$, we round up the length of each slice to the closest $r\frac{\epsilon}{\tilde{n}(1+\epsilon)}|I_\ell|$. Hence, the length of each slice is increased by at most $\frac{\epsilon}{\tilde{n}(1+\epsilon)}|I_\ell|$. Since the number of slices is at most $\tilde{n}$, the total processing time in I_ℓ is increased by at most $\tilde{n}(\frac{\epsilon}{\tilde{n}(1+\epsilon)}|I_\ell|) = \frac{\epsilon}{1+\epsilon}|I_\ell|$, which is the length of the created idle period. Thus, $\mathcal{S}$ is a feasible schedule, and the lemma follows. □

Lemma 5.7. *There is a feasible schedule with energy consumption at most* $(1+\epsilon)^{\alpha_{max}}(1+\frac{2}{\tilde{n}-2})^{\alpha_{max}}(1+\frac{\epsilon}{1-\epsilon})^{\alpha_{max}} \cdot OPT$ *and for each operation* $O_{j,k}$, $J_j \in \mathcal{J}$ *and* $1 \leq k \leq n_j$, *there are two time points* $b_{j,k}$ *and* $c_{j,k}$ *as defined in Lemma 5.6 such that each piece of* $O_{j,k}$ *starts and ends at a time point* $b_{j,k} + h\frac{\epsilon}{\tilde{n}^3}(c_{j,k} - b_{j,k})$ *in* $(b_{j,k}, c_{j,k}]$, *where* $h \geq 0$ *is an integer.*

Proof. Consider a schedule $\mathcal{S}$ satisfying Lemma 5.6. According to this, we have partitioned the interval I_ℓ, $1 \leq \ell \leq \tau$, into polynomial to $\tilde{n}$ and to $1/\epsilon$ number of equal length slots. In each of these slots, each operation $O_{j,k}$, $J_j \in \mathcal{J}$ and $1 \leq k \leq n_j$, either is executed during the whole slot or is not executed at all. Let $b_{j,k}$ and $c_{j,k}$ be the starting time of the first piece and the completion time of the last piece, respectively, of $O_{j,k}$ in $\mathcal{S}$.

We will first transform the schedule $\mathcal{S}$ to a feasible schedule $\mathcal{S}'$ in which the execution time of each operation $O_{j,k}$, $J_j \in \mathcal{J}$ and $1 \leq k \leq n_j$, is at least $\frac{\epsilon}{\tilde{n}}(c_{j,k} - b_{j,k})$ as follows. For each slot s of Lemma 5.6 we increase the processors' speeds so as to create an idle period of length $\epsilon|s|$, where $|s|$ is the length of the slot. This can be done by increasing the speeds by a factor of $1+\frac{\epsilon}{1-\epsilon}$, and hence the total energy consumption in $\mathcal{S}$ is increased

by a factor of $(1+\frac{\epsilon}{1-\epsilon})^\alpha$. For each operation $O_{j,k}$, $J_j \in \mathcal{J}$ and $1 \leq k \leq n_j$, we reserve an $\frac{\epsilon|s|}{\tilde{n}}$ period to each slot s in $(b_{j,k}, c_{j,k}]$. In $\mathcal{S}'$, we decrease the speed of $O_{j,k}$ so that its total work is executed during the periods where $O_{j,k}$ was executed in $\mathcal{S}$ and the additional $c_{j,k} - b_{j,k}$ reserved periods. Therefore, in the final schedule the processing time of each operation $O_{j,k}$ is at least $\frac{\epsilon}{\tilde{n}}(c_{j,k} - b_{j,k})$. After this transformation we apply the Earliest Deadline First (EDF) policy to the operations of each processor separately, considering as release date and deadline of each operation $O_{j,k}$, $J_j \in \mathcal{J}$ and $1 \leq k \leq n_j$, the time points $b_{j,k}$ and $c_{j,k}$, respectively. This ensures that we have a feasible schedule with at most $\tilde{n}$ preemptions, as in EDF an operation may be interrupted only when another operation is released.

Next, we transform $\mathcal{S}'$ to a new schedule $\mathcal{S}''$ in order to satisfy the statement of the lemma. For each operation $O_{j,k}$, $J_j \in \mathcal{J}$ and $1 \leq k \leq n_j$, we split the interval $(b_{j,k}, c_{j,k}]$ into slots of length $\frac{\epsilon}{\tilde{n}^3}(c_{j,k} - b_{j,k})$, i.e., we partition $(b_{j,k}, c_{j,k}]$ into intervals of the form $(b_{j,k}+h\frac{\epsilon}{\tilde{n}^3}(c_{j,k}-b_{j,k}), b_{j,k}+(h+1)\frac{\epsilon}{\mu^3}(c_{j,k}-b_{j,k})]$, where $h \geq 0$ is an integer. As the processing time of J_j in $\mathcal{S}$ is at least $\frac{\epsilon}{\tilde{n}}(c_{j,k} - b_{j,k})$, the execution of $O_{j,k}$ has been partitioned into at least $\tilde{n}^2$ slots. In each of these slots, the operation $O_{j,k}$ either is executed during the whole slot or is executed into a fraction of it. As we have applied the EDF policy, each operation is preempted at most $\tilde{n}$ times, and hence at most $2\tilde{n}$ of these slots are not fully occupied by $O_{j,k}$, since for each preempted piece of $O_{j,k}$ at most two slots may not be completely covered by it. We can modify the schedule $\mathcal{S}'$ and get the schedule $\mathcal{S}''$ in which the operation $O_{j,k}$ is executed only to the slots where it was entirely executed in $\mathcal{S}'$. The number of these slots is at least $\tilde{n}^2 - 2\tilde{n}$. Thus, we have to increase the speed of $O_{j,k}$ by a factor of $1 + \frac{2}{\tilde{n}-2}$, and hence the energy is increased by a factor of $(1 + \frac{2}{\tilde{n}-2})^\alpha$. By taking into account Lemma 5.6 and the fact that $\mathcal{S}''$ is a factor of $(1+\frac{\epsilon}{1-\epsilon})^\alpha$ far from the optimal, the lemma follows. □

Henceforth, we consider schedules that satisfy the above lemma. According to this, we consider that for each operation $O_{j,k}$ there are some time points $b_{j,k}$ and $c_{j,k}$, as defined in Lemma 5.6, such that the interval $(b_{j,k}, c_{j,k}]$ is partitioned into polynomial to $\tilde{n}$ and to $1/\epsilon$ number of equal length slots. In each of these slots, the operation $O_{j,k}$ either is executed during the whole slot or is not executed at all. Moreover, $O_{j,k}$ is executed entirely during $(b_{j,k}, c_{j,k}]$.

We formulate now our problem as an integer program using the idea of configurations as in Section 4.2. Here, a configuration c is a schedule for a single job, that is a feasible schedule for all its operations. Specifically, a configuration determines the time points, with respect to Lemma 5.6, and the slots, with respect to Lemma 5.7, during which each operation of one job is executed. Let $\mathcal{C}_j$ be the set of all possible feasible configurations for job $j \in \mathcal{J}$.

Moreover, in order to ensure the feasibility we merge the slots for all operations as in Section 4.2. Specifically, given a processor $P_i \in \mathcal{P}$, consider the time points of all operations of the form $b_{j,k} + h\frac{\epsilon}{\tilde{n}^3}(c_{j,k} - b_{j,k})$ as introduced in Lemmas 5.6 and 5.7. Let $t_{i,1}, t_{i,2}, \ldots, t_{i,\ell_i}$ be the ordered sequence of these time points. Consider now the intervals $(t_{i,p}, t_{i,p+1}]$, $1 \leq p \leq \ell_i - 1$. In a schedule that satisfies Lemmas 5.6 and 5.7, in each such interval either there is exactly one operation that is executed during the whole interval or the interval is idle. Note also that these intervals might not have the same length. Let

$\mathcal{I}$ be the set of all these intervals for all processors. According to Lemmas 5.6 and 5.7, the size of $\mathcal{I}$ is polynomial to the size of the instance and to $1/\epsilon$.

Note that, given the configuration according to which the job J_j is executed, we can compute the energy consumption $E_{j,c}$ for the execution of J_j. For notational convenience, we say that $I \in (j,c)$, if the interval $I \in \mathcal{I}$ is included in the configuration c for an operation of the job $J_j \in \mathcal{J}$. That is, there is an operation $O_{j,k}$, two time points $b_{j,k}$ and $c_{j,k}$, and a slot $(b_{j,k} + h\frac{\epsilon}{\mu^3}(c_{j,k} - b_{j,k}), b_{j,k} + (h+1)\frac{\epsilon}{\mu^3}(c_{j,k} - b_{j,k})]$ in c that contains I.

$$\min \sum_{J_j \in \mathcal{J}} \sum_{c \in \mathcal{C}_j} E_{j,c} \cdot x_{j,c}$$

$$\sum_{c \in \mathcal{C}_j} x_{j,c} \geq 1 \qquad \forall J_j \in \mathcal{J} \tag{5.8}$$

$$\sum_{c : I \in (j,c)} x_{j,c} \leq 1 \qquad \forall I \in \mathcal{I} \tag{5.9}$$

$$x_{j,c} \in \{0,1\} \qquad \forall J_j \in \mathcal{J}, c \in \mathcal{C}_j \tag{5.10}$$

Constraints (5.8) enforce that each job is entirely executed according to exactly one configuration. Constraints (5.9) ensure that at most one job is executed in each interval $I \in \mathcal{I}$. We consider the fractional relaxation of the above integer program where the integrality constraints $x_{j,c} \in \{0,1\}$ are replaced by the constraints $x_{j,c} \geq 0$, for all $J_j \in \mathcal{J}$ and $c \in \mathcal{C}_j$. This relaxation contains an exponential number of variables, while the number of constraints is polynomial to the size of the instance and to $1/\epsilon$. In order to solve it in polynomial time, we follow the same technique as in Section 4.1. Specifically, we show how to apply the Ellipsoid algorithm for the dual program, by defining a polynomial-time separation oracle for it. The dual of the fractional relaxation is the following.

$$\min \sum_{J_j \in \mathcal{J}} \lambda_j - \sum_{I \in \mathcal{I}} \mu_I$$

$$\lambda_j - \sum_{I \in (j,c)} \mu_I \leq E_{j,c} \qquad \forall J_j \in \mathcal{J}, c \in \mathcal{C}_j \tag{5.11}$$

$$\lambda_j, \mu_I \geq 0 \qquad \forall J_j \in \mathcal{J}, c \in \mathcal{C}_j \tag{5.12}$$

Assume that we are given a solution (λ_j, μ_I) for the dual LP. The separation oracle work as follows. For each job $J_j \in \mathcal{J}$, we try to minimize the term $E_{j,c} + \sum_{I \in (j,c)} \mu_I$. If the value $\min_c\{E_{j,c} + \sum_{I \in (j,c)} \mu_I\}$ is less than λ_j, then we have a violated constraint. Otherwise, if there is no violated constraint for any J_j, then the solution is feasible.

In order to find the configuration that minimizes the above expression, we use dynamic programming. Let $A_{k,c}$ be the part of $E_{j,c} + \sum_{I \in (j,c)} \mu_I$ which corresponds to the operations $O_{j,1}, O_{j,2}, \ldots, O_{j,k}$. Let $B_{k,I}$ be the minimum among value $A_{k,c}$ if all the operations $O_{j,1}, O_{j,2}, \ldots, O_{j,k}$ have to be completed at most at the right endpoint of interval I which corresponds to a time-point as defined in Lemma 5.6. Let $C_{k,\ell,I',I}$ be the minimum possible contribution of the operation $O_{k,j}$ to $B_{k,I}$ if it has to be executed during exactly ℓ slots between the right endpoint of I' and the right endpoint of I. Again we assume that the right endpoints of intervals I' and I correspond to time points as defined in Lemma 5.6.

Let $D_{k,I',I} = \min_\ell\{C_{k,\ell,I',I}\}$. For notational convenience, if the interval I' precedes the interval I we write $I' < I$. Then, we consider the following dynamic program

$$B_{k,I} = \min_{I'<I}\{B_{k-1,I'} + D_{k,I',I}\}$$

By definition, the term $D_{k,I',I}$ is the minimum among $C_{k,\ell,I',I}$, for all ℓ. The term $C_{k,\ell,I',I}$ can be computed in polynomial time for any k, ℓ, I' and I. Indeed, as the slots of $O_{j,k}$ are of equal length, the energy consumption using ℓ slots does not depend on the slots that are included to the configuration. Hence, the quantity $E_{j,c}$ is fixed for a given ℓ. Consider now a slot s of $O_{j,k}$ between the right endpoint of I' and the right endpoint of I. We call $\sum_{I\in s}\mu_I$ the value of s. In order to minimize $C_{k,\ell,I',I}$, we have to find the ℓ slots with minimum value $\sum_{I\in s}\mu_I$. As the number of slots is polynomial to the size of the instance and to $1/\epsilon$, this can be done in polynomial time.

Thus, for each job $J_j \in \mathcal{J}$, we can compute in polynomial time the minimum $E_{j,c} + \sum_{I\in(j,c)}\mu_I$ among all the configurations $c \in \mathcal{C}_j$, and hence we have a polynomial-time separation oracle. Therefore, we can apply the Ellipsoid algorithm to the dual and obtain a polynomial-time algorithm for the fractional relaxation as in Section 4.2. Then, by applying the same randomized rounding algorithm and analysis as in Section 4.2 we get the following theorem.

Theorem 5.3. *There exists algorithm for the preemptive energy minimization problem in a job shop which is polynomial to $\tilde{n}$ and $1/\epsilon$ and achieves an approximation ratio of $(1+\epsilon)^\alpha(1+\frac{2}{\mu-2})^\alpha(1+\frac{\epsilon}{1-\epsilon})^\alpha\tilde{B}_\alpha$.*

Chapter 6

Temperature-Aware Scheduling

In this chapter, we switch our attention to temperature-aware scheduling and we study multiprocessor problems under the discrete thermal model.

Initially, in Section 6.1, we study the approximability of $\mathbf{T}, \mathbf{P}|\mathbf{p_j} = \mathbf{1}, \mathbf{h_j}|\mathbf{C}_{\max}(\mathbf{\Theta})$ in which the objective is the minimization of the makespan under a threshold on the temperature which must not be exceeded. First, we show that there is no $(\frac{4}{3} - \epsilon)$-approximation algorithm, unless $\mathcal{P} = \mathcal{NP}$, through a reduction from the Numerical 3-Dimensional Matching problem. Then, we propose a constant factor approximation algorithm which is based on a transformation of the problem to the problem $\mathbf{P}||\mathbf{C_{max}}$. Our algorithm uses any ρ-approximation algorithm for $\mathbf{P}||\mathbf{C_{max}}$ as black box and it achieves an approximation ratio of 2ρ. Given that the best known algorithm for the latter problem is a polynomial time approximation scheme (PTAS), we obtain a $(2 + \epsilon)$-approximation algorithm for $\mathbf{T}, \mathbf{P}|\mathbf{p_j} = \mathbf{1}, \mathbf{h_j}|\mathbf{C}_{\max}(\mathbf{\Theta})$. In order to obtain a faster running time, we consider the case of using the LPT (Longest Processing Time first) as a black box algorithm, instead of the PTAS, which is known to be 4/3-approximate for $\mathbf{P}||\mathbf{C_{max}}$. Based on our previous analysis, the approximation ratio of our algorithm is 8/3. However, we improve this ratio to $\frac{7}{3} - \frac{1}{3m}$ through an LPT-oriented analysis.

Subsequently, in Section 6.2, we address problems in which the temperature is the objective function. We begin with the maximum temperature minimization problem $\mathbf{T}, \mathbf{P}|\mathbf{p_j} = \mathbf{1}, \mathbf{h_j}, \mathbf{d_j} = \mathbf{d}|\mathbf{\Theta}_{\max}$ and we present a tight $\frac{4}{3}$-approximation algorithm. Recall that this problem is strongly $\mathcal{NP}$-hard (see [30]). Then, we turn our attention to the problem $\mathbf{T}, \mathbf{P}|\mathbf{p_j} = \mathbf{1}, \mathbf{h_j}, \mathbf{d_j} = \mathbf{d}|\sum \mathbf{\Theta_t}$ of minimizing the average temperature and we show that it can be solved in polynomial time.

6.1 Makespan Minimization

In this section, we study the approximability of the multiprocessor makespan minimization problem $\mathbf{T}, \mathbf{P}|\mathbf{p_j} = \mathbf{1}, \mathbf{h_j}|\mathbf{C}_{\max}(\mathbf{\Theta})$. In this problem, we are given a set of n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ with unit processing times which have to be scheduled by a set of m parallel identical processors $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$. Each job J_j is associated with a heat contribution $h_j \in [0, 2]$. The time horizon can be partitioned into unit-length slots $[0, 1), [1, 2), \ldots, [t, t + 1), \ldots$ such that, at every time slot $[t, t + 1)$, on a given processor $P_i \in \mathcal{P}$, either a single job is executed on P_i during the whole slot or P_i is idle. We

consider the temperatures only at integer times and if the temperature of the processor P_i is Θ_t at time t, then P_i's temperature Θ_{t+1} at time $t+1$ can be computed as follows. If the job $J_j \in \mathcal{J}$ is executed on P_i during the time slot $[t, t+1)$, then

$$\Theta_{t+1} = \frac{\Theta_t + h_j}{2}$$

If P_i is idle during $[t, t+1)$, then

$$\Theta_{t+1} = \frac{\Theta_t}{2}$$

Without loss of generality, we assume that the initial temperature is equal to zero, i.e. $\Theta_0 = 0$. At each integer time t, the temperature of any processor must not exceed a temperature threshold which is equal to one, i.e. it must be the case that $\Theta_t \leq 1$. Our objective is to find a feasible schedule with minimum makespan.

6.1.1 Inapproximability

We start with a negative result on the approximability of our problem. The proof of the next theorem is inspired by the NP-hardness reduction for the throughput maximization problem under the same model which can be found in [30].

Theorem 6.1. *There is no polynomial time algorithm achieving an absolute approximation ratio better than* $4/3$ *for the minimum makespan problem* $\mathbf{T}, \mathbf{P}|\mathbf{p_j} = \mathbf{1}, \mathbf{h_j}|\mathbf{C}_{\max}(\mathbf{\Theta})$*, unless* $\mathcal{P} = \mathcal{NP}$*.*

Proof. We give a reduction from Numerical 3-Dimensional Matching (N3DM) where we are given three sets A, B, C of n integers each and an integer β, and the question is whether $A \cup B \cup C$ can be partitioned into n disjoint triples $(a, b, c) \in A \times B \times C$ such that each triple contains exactly one integer from each set A, B, C and $a + b + c = \beta$ for every such triple. Without loss of generality, we assume that $\sum_{x \in A \cup B \cup C} x = \beta n$ and $x \leq \beta$ for each $x \in A \cup B \cup C$. The N3DM problem is known to be NP-complete (see [35]).

Given an instance I of N3DM, we construct an instance I' of $\mathbf{T}, \mathbf{P}|\mathbf{p_j} = \mathbf{1}, \mathbf{h_j}|\mathbf{C}_{\max}(\mathbf{\Theta})$ consisting of n processors and $3n$ jobs, one for each integer in $A \cup B \cup C$. Considering the function $f(x) = \frac{1}{25}\left(1 + \frac{x}{8\beta}\right)$, we set $h(a) = 8f(a) + 1$ for each $a \in A$, $h(b) = 4f(b) + 1$ for each $b \in B$ and $h(c) = 2f(c) + 1$ for each $c \in C$.

The reduction works by showing that it is hard to decide whether the optimal schedule is of length three or not.

Claim 6.1. *There is a N3DM for the instance I if and only if there is a feasible schedule for the instance I' of $\mathbf{T}, \mathbf{P}|\mathbf{p_j} = \mathbf{1}, \mathbf{h_j}|\mathbf{C}_{\max}(\mathbf{\Theta})$ of length three.*

Proof. ($\Rightarrow$) Assume that there is a solution for N3DM. For the i-th triple (a_i, b_i, c_i), $1 \leq i \leq n$, in this solution, we schedule on the processor P_i the jobs corresponding to a_i, b_i and c_i in the first, second and third slots, respectively. For the temperatures, $\Theta(a_i), \Theta(b_i), \Theta(c_i)$, on the processor P_i after each one of those executions we have

$$\Theta(a_i) = \frac{8f(a_i)+1}{2} \leq \frac{8f(\beta)+1}{2} = \frac{\frac{8}{25}\left(1+\frac{\beta}{8\beta}\right)+1}{2} = \frac{34}{50} \leq 1$$

$$\Theta(b_i) = \frac{8f(a_i)+1}{4} + \frac{4f(b_i)+1}{2} = \frac{3}{4} + 2\left(\frac{1}{25}\left(1+\frac{a_i}{8\beta}\right) + \frac{1}{25}\left(1+\frac{b_i}{8\beta}\right)\right) \leq \frac{3}{4} + \frac{4}{25} + \frac{\beta}{100\beta} = \frac{93}{100} \leq 1$$

$$\Theta(c_i) = \frac{8f(a_i)+1}{8} + \frac{4f(b_i)+1}{4} + \frac{2f(c_i)+1}{2} = \frac{7}{8} + \frac{1}{25}\left(1+\frac{a_i}{8\beta}\right) + \frac{1}{25}\left(1+\frac{b_i}{8\beta}\right) + \frac{1}{25}\left(1+\frac{c_i}{8\beta}\right) = \frac{7}{8} + \frac{3}{25} + \frac{\beta}{200\beta} = 1$$

and hence there is a feasible schedule of length three.

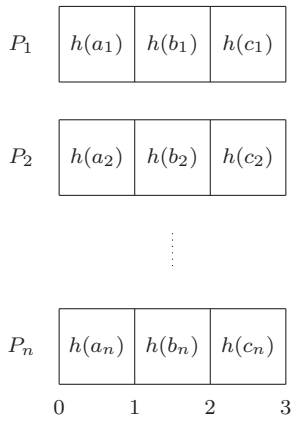


Figure 6.1: The schedule produced from a given N3DM $(a_1, b_1, c_1), (a_2, b_2, c_2), \ldots, (a_n, b_n, c_n)$ of the set A.

($\Leftarrow$) Assume, now, that there is a feasible schedule of length three. In this schedule there are exactly three jobs in each processor, since there are $3n$ jobs in total.

If a job corresponding to an integer $a \in A$ is scheduled to the second slot of a processor, then the temperature threshold $\Theta = 1$ is violated after the third slot of this processor. Indeed the temperature at this slot will be at least

$$\frac{2f(0)+1}{8} + \frac{8f(0)+1}{4} + \frac{2f(0)+1}{2} = \frac{7}{8} + \frac{1}{25}\left(1+\frac{0}{8\beta}\right)\left(\frac{2}{8}+\frac{8}{4}+\frac{2}{2}\right) = \frac{201}{200} > 1.$$

In a similar way, we can show that a job corresponding to an integer $a \in A$ cannot be scheduled to the third slot of a processor:

$$\frac{2f(0)+1}{8} + \frac{2f(0)+1}{4} + \frac{8f(0)+1}{2} = \frac{7}{8} + \frac{1}{25}\left(1+\frac{0}{8\beta}\right)\left(\frac{2}{8}+\frac{2}{4}+\frac{8}{2}\right) = \frac{213}{200} > 1.$$

Hence, each of the n jobs corresponding to one of the n integers $a \in A$ is scheduled to the first slot of a processor. Moreover, we can show that a job corresponding to an integer $b \in B$ cannot be scheduled to the third slot of a processor:

$$\frac{8f(0)+1}{8} + \frac{2f(0)+1}{4} + \frac{4f(0)+1}{2} = \frac{7}{8} + \frac{1}{25}\left(1+\frac{0}{8\beta}\right)\left(\frac{8}{8}+\frac{2}{4}+\frac{4}{2}\right) = \frac{203}{200} > 1.$$

In all, in each processor exactly three jobs are scheduled: a job $a \in A$ in the first slot, a job $b \in B$ in the second slot, and a job $c \in C$ in the third slot. Therefore, the jobs of a processor correspond to a feasible triple for N3DM.

To finish our proof, we have to show that each triple sums up to β. If this does not hold then there is a triple (a, b, c) for which $a + b + c > \beta$, since $\sum_{x \in A \cup B \cup C} x = \beta n$. The

temperature of the third slot of the processor in which the corresponding jobs to this triple are scheduled is

$$\frac{8f(a)+1}{8} + \frac{4f(b)+1}{4} + \frac{2f(c)+1}{2} = \frac{7}{8} + \frac{1}{25}\left(3 + \frac{a+b+c}{8\beta}\right) > \frac{7}{8} + \frac{1}{25}\left(3 + \frac{\beta}{8\beta}\right) = 1,$$

which is a contradiction that there is a feasible schedule. □

This completes the proof of Theorem 6.1 since an approximation ratio better than 4/3 would be able to decide the problem N3DM. □

Note that the result of Theorem 6.1 allows the possibility of an asymptotic PTAS or even an additive constant approximation ratio.

6.1.2 Approximation Algorithm based on a transformation to $P||C_{max}$

In what follows, we present an approximation algorithm for $\mathbf{T}, \mathbf{P}|\mathbf{p_j} = \mathbf{1}, \mathbf{h_j}|\mathbf{C_{max}}(\mathbf{\Theta})$ which is based on a transformation of the instance to an instance of the problem $\mathbf{P}||\mathbf{C_{max}}$.

Note that, in order to respect the temperature threshold, a schedule may have to contain idle slots. To argue about the number of idle slots that are needed before the execution of each job, we will introduce first an appropriate partition of the set of jobs according to their heat contribution. In particular, for each integer $k \geq 0$, we can argue separately for jobs whose heat contribution belongs to the interval $(2-\frac{1}{2^{k-1}}, 2-\frac{1}{2^k}]$; recall that $h_j \leq 2$, for $J_j \in \mathcal{J}$. Moreover, the interval to which a job of heat contribution h_j belongs to is indexed by k_j, that is

$$k_j = \max\{k \in \mathbb{N} \mid h_j > 2 - \frac{1}{2^{k-1}}\}$$

Our algorithm and its analysis are based on the following proposition for the structure of any feasible schedule.

Lemma 6.1.
(i) Let $\mathcal{J}'$ be the set of jobs of heat contribution $h_j > 1$; $|\mathcal{J}'| = n'$. Any feasible schedule can be transformed into another feasible one of at most the same length where exactly $\min\{n', m\}$ jobs in $\mathcal{J}'$ are executed in the first slot of the processors.
(ii) Any schedule where every J_j is executed right after k_j consecutive idle slots is feasible.
(iii) In an optimal schedule, if a job $J_{j'}$ is executed before a job J_j on the same processor, where $h_{j'}, h_j > 1$, then there are at least $k_j - 1$ slots between $J_{j'}$ and J_j, which are either idle or execute jobs of heat contribution at most one.

Proof.
(i) Consider a feasible schedule that has less than $\min\{n', m\}$ jobs in $\mathcal{J}'$ executed in the first slot of the processors.

Assume, first, that in this schedule there is a processor $P_i \in \mathcal{P}$ in which a job $J_j \in \mathcal{J} \setminus \mathcal{J}'$ is executed in its first slot and there is at least one job of $\mathcal{J}'$ executed on P_i. Let $J_{j'} \in \mathcal{J}'$ be the earliest of these jobs which is executed in slot $t > 1$. By swapping the jobs J_j and $J_{j'}$, the temperature Θ'_t of processor P_i after slot t is decreased. Indeed, let Θ_t be the temperature of processor P_i after slot t and Θ' be the contribution of jobs

executed in slots $2, 3, \ldots, t-1$ to Θ_t, that is $\Theta_t = \frac{h_j}{2^t} + \Theta' + \frac{h_{j'}}{2}$. After the swap it holds that $\Theta'_t = \frac{h_{j'}}{2^t} + \Theta' + \frac{h_j}{2} < \Theta_t$, since $h_j < h_{j'}$. Thus, the temperature of any slot $t' \geq t$ in P_i is decreased. Moreover, by assumption, each slot t', $2 \leq t' \leq t-1$, of P_i executes a job in $\mathcal{J} \setminus \mathcal{J}'$. Hence, no new idle slots are required for these jobs, although the temperature before their execution is increased. Therefore, the new schedule is feasible and it has the same length.

If there is not such a processor, then let $J_j \in \mathcal{J} \setminus \mathcal{J}'$ be a job executed in the first slot of some processor P_i and $J_{j'} \in \mathcal{J}'$ be a job executed in t-th, $t > 1$, slot of processor $P_{i'}$. By swapping the jobs J_j and $J_{j'}$ the temperature of any slot $t' \geq t$ of processor $P_{i'}$ is decreased as $h_j < h_{j'}$. Moreover, by assumption, the processor P_i contains only jobs in $\mathcal{J} \setminus \mathcal{J}'$, and, as in the previous case, no new idle slots are required for these jobs. Therefore, after the swap we get a feasible schedule of the same length.

(ii) Consider a schedule that is feasible up until the execution of the job preceding J_j. Let x be the number of idle slots before the execution of job J_j and let Θ' be the temperature of the processor before the first of these x slots. Since the schedule is feasible before J_j, we have that $\Theta' \leq 1$. The temperature will become $\frac{\Theta'}{2^x}$, after the last idle slot, and $\frac{\frac{\Theta'}{2^x}+h_j}{2}$ after the execution of job J_j. For such a schedule to be feasible we need that $\frac{\frac{\Theta'}{2^x}+h_j}{2} \leq 1$, that is, $2^x \geq \frac{\Theta'}{2-h_j}$. Since $h_j \leq \frac{2^{k_j+1}-1}{2^{k_j}}$, it follows that $\frac{\Theta'}{2-h_j} \leq \frac{1}{2-\frac{2^{k_j+1}-1}{2^{k_j}}} = 2^{k_j}$. This means that with at least k_j idle slots, feasibility is ensured.

(iii) Let Θ_t be the temperature of the processor before executing $J_{j'}$. Next, after the execution of $J_{j'}$ we have $\Theta_{t+1} = \frac{\Theta_t + h_{j'}}{2}$. Then, after x slots (idles or executing jobs of heat contribution $h \leq 1$) we get a temperature $\Theta_{t+x+1} \geq \frac{\Theta_t+h_{j'}}{2} \cdot \frac{1}{2^x}$. In order for J_j to be executed in the next slot, it should hold that $\Theta_{t+x+1} + h_j \leq 2$, that is $2^x \geq \frac{\Theta_t+h_{j'}}{2(2-h_j)}$. Since, $\Theta_t \geq 0$, $h_{j'} > 1$ and $h_j > \frac{2^{k_{j'}}-1}{2^{k_{j'}-1}}$ we get $2^x \geq \frac{\Theta_t+h_{j'}}{2(2-h_j)} > \frac{1}{2(2-\frac{2^{k_j}-1}{2^{k_j-1}})} = \frac{1}{\frac{2}{2^{k_j-1}}} = 2^{k_j-2}$, that is $x \geq k_j - 1$. □

In what follows we consider instances with $n > m$, for otherwise the problem becomes trivial. By Lemma 6.1(i), we also assume that the number of jobs of heat contribution $h_i > 1$ is greater than m. If this is not the case, all jobs can be executed without any idle slot before them and the length of an optimal schedule is exactly $\lceil \frac{n}{m} \rceil$. We consider the jobs in non-increasing order of their heat contributions, i.e., $h_1 \geq h_2 \geq \ldots \geq h_n$, and we define $\mathcal{A} = \{J_1, J_2, \ldots, J_m\}$ and $\mathcal{B} = \{J_{m+1}, J_{m+2}, \ldots, J_n\}$. Our algorithm schedules first the jobs in $\mathcal{A}$ to the first slot of each processor. Each one of the jobs in $\mathcal{B}$ is scheduled by leaving before its execution *exactly* k_j idle slots, according to the Lemma 6.1(ii). In this way, our problem, for the jobs in $\mathcal{B}$, is transformed to an instance of the classical makespan problem on parallel machines, $\mathbf{P}||\mathbf{C}_{\max}$, where the processing time of each job is $p_j = k_j + 1$, that is k_j idle slots plus its original unit processing time. Then, these jobs are scheduled using any known approximation algorithm for $\mathbf{P}||\mathbf{C}_{\max}$. A pseudocode of our algorithm is given in Algorithm 6.1.

From now on we fix an instance of our problem and we denote by SOL the length of the schedule $\mathcal{S}$ provided by Algorithm 6.1 and by OPT the length of an optimal schedule

$\mathcal{S}^*$ for our original scheduling problem. For the presentation and the analysis of our algorithm, we denote by $\mathcal{I}_B$ and $\mathcal{I}_B^+$ the instances of $\mathbf{P}||\mathbf{C}_{\max}$ consisting only of jobs in $\mathcal{B}$ with processing times $p_j = k_j$ and $p_j = k_j + 1$, respectively, for each $J_j \in \mathcal{B}$.

Algorithm 6.1

1: Sort the jobs so that $h_1 \geq h_2 \geq \ldots \geq h_n$.
2: Let $\mathcal{A} = \{J_1, J_2, \ldots, J_m\}$, and $\mathcal{B} = \{J_{m+1}, J_{m+2}, \ldots, J_n\}$.
3: Schedule each job $J_j \in \mathcal{A}$ to the first slot of processor $P_j \in \mathcal{P}$.
4: Run an algorithm $\mathcal{R}$ of $\mathbf{P}||\mathbf{C}_{\max}$ for the instance $\mathcal{I}_B^+$.

For an instance $\mathcal{I}$ of $\mathbf{P}||\mathbf{C}_{\max}$, we denote by $\mathcal{S}(\mathcal{I})$ the schedule found by an algorithm $\mathcal{R}$ and by $\mathcal{C}(\mathcal{I})$ the length of this schedule. In a similar way, we denote by $\mathcal{S}^*(\mathcal{I})$ and $\mathcal{C}^*(\mathcal{I})$ an optimal schedule for $\mathbf{P}||\mathbf{C}_{\max}$ and the length of this optimal schedule, respectively. Clearly, $SOL = 1 + \mathcal{C}(\mathcal{I}_B^+)$. To analyze the Algorithm 6.1, we need a lower bound on the optimal makespan. To derive this bound we will utilize an optimal schedule $\mathcal{S}^*(\mathcal{I}_B)$. Note that for jobs with $h_j \in (0, 1]$, $k_j = 0$, hence the schedule $\mathcal{S}^*(\mathcal{I}_B)$ involves only jobs for which $h_j > 1$.

Lemma 6.2. *For the optimal makespan it holds that*

$$OPT \geq \max\{\frac{n}{m}, 1 + \mathcal{C}^*(\mathcal{I}_B)\}$$

Proof. The first bound on the optimal makespan follows trivially by considering all jobs requiring a single slot for their execution.

For the second bound, let $\mathcal{A}^*$, $|\mathcal{A}^*| = m$, be the set of jobs executed in the first slot of the m processors in an optimal solution and $\mathcal{B}^* = \mathcal{J} \setminus \mathcal{A}^*$.

Consider, first, an auxiliary schedule of length OPT^-, identical to the optimal apart from the fact that each job in $\mathcal{B}^* \cap \mathcal{A}$ has been replaced by a different job in $\mathcal{A}^* \cap \mathcal{B}$. Observe that in this schedule, the jobs executed in the first slot of the processors remain $\mathcal{A}^*$ while the jobs executed in the remaining slots are the jobs in $\mathcal{B}$. Since each job in $\mathcal{B}$ has smaller or equal heat contribution than any job in $\mathcal{A}$, it follows that $OPT \geq OPT^-$.

Consider, next, the schedule $\mathcal{S}^*(\mathcal{I}_B)$. For this schedule it holds that, $OPT^- \geq 1 + \mathcal{C}^*(\mathcal{I}_B)$, since by Lemma 6.1(i),(iii) each job in $\mathcal{B}$ requires at least k_j slots to be executed; recall that we consider instances where the number of jobs of heat contribution $h_j > 1$ is greater than m and that jobs in $\mathcal{B}$ with $h_j \leq 1$, and hence $k_j = 0$, do not appear in the schedule $\mathcal{S}^*(\mathcal{I}_B)$. □

It is well-known that the $\mathbf{P}||\mathbf{C}_{\max}$ problem is strongly NP-hard and a series of constant approximation algorithms and PTASs have been proposed, e.g. [43]. Our main result in this section is that in step 4 of Algorithm 6.1 we can use any algorithm $\mathcal{R}$ for $\mathbf{P}||\mathbf{C}_{\max}$ to obtain twice the approximation ratio of $\mathcal{R}$ for our problem.

Theorem 6.2. *Algorithm 6.1 is 2ρ-approximate ratio for $\mathbf{T}, \mathbf{P}|\mathbf{p_i} = \mathbf{1}, \mathbf{h_i}|\mathbf{C}_{\max}(\boldsymbol{\Theta})$, where ρ is the approximation ratio of the algorithm $\mathcal{R}$ for $\mathbf{P}||\mathbf{C}_{\max}$.*

Proof. A ρ-approximation algorithm $\mathcal{R}$ implies that $\frac{\mathcal{C}(\mathcal{I}_B^+)}{\mathcal{C}^*(\mathcal{I}_B^+)} \leq \rho$. Hence, $SOL = 1 + \mathcal{C}(\mathcal{I}_B^+) \leq 1 + \rho \cdot \mathcal{C}^*(\mathcal{I}_B^+)$.

To obtain an upper bound to $\mathcal{C}^*(\mathcal{I}_B^+)$ we start from the schedule $\mathcal{S}^*(\mathcal{I}_B)$. The processing times of jobs in the latter schedule are reduced by one with respect to the former one, and the jobs in $\mathcal{B}$ with $h_j \leq 1$ do not appear in schedule $\mathcal{S}^*(\mathcal{I}_B)$. Let $\mathcal{B}' \subseteq \mathcal{B}$ be this set of jobs.

We transform the schedule $\mathcal{S}^*(\mathcal{I}_B)$ to a new schedule $\mathcal{S}'(\mathcal{I}_B^+)$ in two successive steps: (i) we increase the processing time of jobs in $\mathcal{B} \setminus \mathcal{B}'$ from k_j to $k_j + 1$, and (ii) we introduce the jobs in $\mathcal{B}'$ with unit processing time, at the end of the resulting schedule in a first-fit manner. Clearly, for the length, $\mathcal{C}'(\mathcal{I}_B^+)$, of this new schedule it holds that $\mathcal{C}^*(\mathcal{I}_B^+) \leq \mathcal{C}'(\mathcal{I}_B^+)$ as both of them refer to the same instance $\mathcal{I}_B^+$. Let us now bound $\mathcal{C}'(\mathcal{I}_B^+)$ in terms of $\mathcal{C}^*(\mathcal{I}_B)$.

If $\mathcal{C}'(\mathcal{I}_B^+) \leq 2\mathcal{C}^*(\mathcal{I}_B)$, then $\frac{SOL}{OPT} \leq \frac{1+2\rho\mathcal{C}^*(\mathcal{I}_B)}{1+\mathcal{C}^*(\mathcal{I}_B)} \leq 2\rho$, since $\rho \geq 1$.

If $\mathcal{C}'(\mathcal{I}_B^+) > 2\mathcal{C}^*(\mathcal{I}_B)$, then we consider the construction of $\mathcal{S}'(\mathcal{I}_B^+)$ and we argue about the completion time of a critical processor in $\mathcal{S}^*(\mathcal{I}_B)$, i.e., the processor that finishes last. By step (i), the length of schedule $\mathcal{S}^*(\mathcal{I}_B)$ increases at most twice, since each job in $\mathcal{B} \setminus \mathcal{B}'$ has processing time at least one and this is increased by 1. As $\mathcal{C}'(\mathcal{I}_B^+) > 2\mathcal{C}^*(\mathcal{I}_B)$, in the last slot of $\mathcal{S}'(\mathcal{I}_B^+)$ all non-idle processors execute jobs of $\mathcal{B}'$. By step (ii), all but the last time slots of $\mathcal{S}'(\mathcal{I}_B^+)$ are busy. Hence, the critical processor in $\mathcal{S}^*(\mathcal{I}_B)$ finishes in $\mathcal{S}'(\mathcal{I}_B^+)$ the earliest at time $\mathcal{C}'(\mathcal{I}_B^+) - 1$. Moreover, this processor is assigned the minimum total increase at the end of the transformation, since it finishes last in $\mathcal{S}^*(\mathcal{I}_B)$. As the total increase of the processing times from $\mathcal{S}^*(\mathcal{I}_B)$ to $\mathcal{S}'(\mathcal{I}_B^+)$ is $n-m$, it follows that the length of the critical processor increases at most by $\frac{n-m}{m}$. Hence, $\mathcal{C}'(\mathcal{I}_B^+) - 1 \leq \mathcal{C}^*(\mathcal{I}_B) + \frac{n-m}{m}$, that is $\mathcal{C}'(\mathcal{I}_B^+) \leq \mathcal{C}^*(\mathcal{I}_B) + \frac{n}{m}$. Thus, by Lemma 6.2 we get $\frac{SOL}{OPT} \leq \frac{1+\rho(\mathcal{C}^*(\mathcal{I}_B)+\frac{n}{m})}{\max\left\{\frac{n}{m}, 1+\mathcal{C}^*(\mathcal{I}_B)\right\}} \leq \frac{1+\rho\mathcal{C}^*(\mathcal{I}_B)}{1+\mathcal{C}^*(\mathcal{I}_B)} + \frac{\rho\frac{n}{m}}{\frac{n}{m}} \leq 2\rho$. □

For the case of a single processor the $\mathbf{1}||\mathbf{C_{max}}$ problem is trivially polynomial, whereas for multiple processors there are well known PTAS's, e.g. [43]. Hence the main implication of Theorem 6.2 is:

Corollary 6.1. *For any $\epsilon > 0$, there is a $(2+\epsilon)$-approximation algorithm for the problem $\mathbf{T}, \mathbf{P}|\mathbf{p_j} = \mathbf{1}, \mathbf{h_j}|\mathbf{C}_{\max}(\mathbf{\Theta})$. For the case of a single processor, there is an algorithm that achieves an approximation ratio of 2.*

6.1.3 LPT oriented Approximation Algorithm

To obtain the ratio of $2 + \epsilon$, as stated above, one needs to use a PTAS for the classical makespan problem in step 4 of Algorithm 6.1, resulting in a running time that is exponential in $1/\epsilon$. To achieve more practical running times, we can investigate the use of other algorithms for step 4. In particular, if the standard Longest Processing Time (LPT) algorithm is used, then Theorem 6.2 leads to a $2(\frac{4}{3} - \frac{1}{3m})$ approximation ratio within $O(n \log n)$ time. Recall that the LPT algorithm greedily assigns the next job (in non-increasing order of their processing times) to the first available processor [38]. In the next theorem we are able to improve this ratio to 7/3, based on an LPT oriented analysis of Algorithm 6.1.

Theorem 6.3. *Algorithm 6.1 using the LPT rule in step 4 achieves an approximation ratio of* $\frac{7}{3} - \frac{1}{3m}$ *for* $\mathbf{T}, \mathbf{P}|\mathbf{p_i} = \mathbf{1}, \mathbf{h_i}|\mathbf{C}_{\max}(\mathbf{\Theta})$ *within* $O(n \log n)$ *time.*

Proof. Our proof follows the standard analysis given in [38], for the classical multiprocessor scheduling problem. For the lower bound on the length of an optimal schedule we use Lemma 6.2 and the fact that $\mathcal{C}^*(\mathcal{I}_B) \geq \frac{\sum_{j=m+1}^{n} k_j}{m}$. Hence, $OPT \geq \max\{\frac{n}{m}, 1 + \frac{\sum_{j=m+1}^{n} k_j}{m}\}$, and by the standard average argument we get

$$OPT \geq \frac{m + \sum_{j=m+1}^{n} k_j + n}{2m} = 1 + \frac{\sum_{j=m+1}^{n} (k_j + 1)}{2m}.$$

To upper bound the length SOL of the schedule $\mathcal{S}$ returned by Algorithm 6.1 we consider the job J_ℓ which finishes last in $\mathcal{S}$. Clearly $\ell > m$, for otherwise there are at most m jobs to be scheduled and the problem becomes trivial.

The job J_ℓ will start being executed not later than $1 + \frac{\sum_{j=m+1, j\neq\ell}^{n} (k_j+1)}{m}$, and hence, it holds that

$$SOL \leq 1 + \frac{\sum_{j=m+1, j\neq\ell}^{n} (k_j+1)}{m} + (k_\ell + 1) = 1 + \frac{\sum_{j=m+1}^{n} (k_j+1)}{m} + \left(1 - \frac{1}{m}\right)(k_\ell + 1).$$

Thus, we get $SOL \leq 2OPT - 1 + \left(1 - \frac{1}{m}\right)(k_\ell + 1)$.

If $k_\ell \leq OPT/3$, then the theorem follows directly.

If $k_\ell > OPT/3$, then we consider the subinstance, I', of the original problem that contains only the jobs of heat contribution at least h_ℓ, i.e., $J' = \{J_1, J_2, \ldots, J_\ell\}$. Obviously, $k_1 \geq k_2 \geq \ldots \geq k_\ell > \frac{OPT}{3}$ and $k_\ell \geq 1$, as k_ℓ is an integer. Moreover, for the length of an optimal schedule, $C^*(I')$, of the subinstance I' it holds that $C^*(I') \leq OPT$. As $\ell > m$, the lengths of the schedules returned by Algorithm 6.1 for instances I and I' are equal, i.e., $C(I') = SOL$. Hence, $\frac{SOL}{OPT} \leq \frac{C(I')}{C^*(I')}$.

In an optimal schedule of I' there are at most three jobs in each processor, for otherwise, if there is a processor with four assigned jobs, the length of that schedule will be, by Lemma 6.1(iii), at least $1 + 3k_\ell > OPT$, a contradiction. Hence, $\ell \leq 3m$.

Algorithm 6.1 schedules the jobs of I' as follows: the job J_j, $1 \leq j \leq m$, is scheduled to the first slot of processor P_j, the job J_{m+j}, $1 \leq j \leq m$, to the $(1 + (k_{m+j} + 1))$-th slot of processor P_j and job J_{2m+j}, $1 \leq j \leq m$, accordingly to the LPT rule.

If $m < \ell \leq 2m$, then the length of the above schedule is $C(I') = 1 + (k_{m+1} + 1) = 2 + k_{m+1}$. By Lemma 6.2 it follows that $C^*(I') \geq 1 + k_{m+1}$, since there is a processor executing at least two jobs in $\{J_1, J_2, \ldots, J_{m+1}\}$. Hence, $\frac{SOL}{OPT} \leq \frac{C(I')}{C^*(I')} \leq \frac{2+k_{m+1}}{1+k_{m+1}} \leq \frac{3}{2}$, as $k_{m+1} \geq k_\ell \geq 1$.

If $2m < \ell \leq 3m$, then the Algorithm 6.1 schedules in the first processor either the jobs J_1 and J_{m+1} or the jobs J_1, J_{m+1} and J_ℓ. In the first case, the job J_ℓ starts its execution not later than the slot $1 + (k_{m+1} + 1)$, for otherwise J_ℓ would have been scheduled by Algorithm 6.1 in processor P_1, that is $C(I') \leq 1 + (k_{m+1} + 1) + (k_\ell + 1)$. In the second case, J_ℓ is the job that finishes last, that is $C(I') = 1 + (k_{m+1} + 1) + (k_\ell + 1)$. Thus, in both cases it holds that $C(I') \leq 3 + k_{m+1} + k_\ell$.

For an optimal schedule for I', Lemma 6.2 implies as before that $C^*(I') \geq 1 + k_{m+1}$. Moreover, in such a schedule there is a processor with at least three jobs, and hence $C^*(I') \geq 1 + 2k_\ell$. Combining these two bounds we get $C^*(I') \geq 1 + \frac{k_{m+1}}{2} + k_\ell$.

Therefore, we get $\frac{SOL}{OPT} \leq \frac{C(I')}{C^*(I')} \leq \frac{6+2k_{m+1}+2k_\ell}{2+k_{m+1}+2k_\ell}$. This ratio is decreasing with k_ℓ and as $k_\ell \geq 1$ we get $\frac{SOL}{OPT} \leq \frac{8+2k_{m+1}}{4+k_{m+1}} = 2$, and the proof is completed. □

Note that the $\left(\frac{4}{3} - \frac{1}{3m}\right)$-approximation ratio of the LPT algorithm for the classical makespan problem on parallel machines is tight. Concerning the tightness of our algorithm, we are able to give an instance where it achieves a 2-approximation ratio. This instance consists of $m(k+2)$ jobs: a set $\mathcal{A}$ of m jobs of heat contribution $h_j = 2$, a set $\mathcal{B}$ of m jobs of heat contribution $h_j = 2 - \frac{3}{2^{k+1}}$, and a set $\mathcal{C}$ of mk jobs of heat contribution $h_j = \frac{1}{2(2^k-1)}$.

An optimal solution for this instance is to schedule the jobs in the following way: every processor executes a job of $\mathcal{A}$ in the first slot, k jobs of $\mathcal{C}$ in slots $2, 3, \ldots, k+1$, and a job of $\mathcal{B}$ in slot $k+2$. The temperature of every processor after slot $k+1$ is $\frac{1}{2^k} + \frac{1}{2(2^k-1)} \cdot \frac{2^k-1}{2^k} = \frac{3}{2^{k+1}}$, and hence a job of $\mathcal{B}$ can be executed in slot $k+2$. Moreover, as the jobs of $\mathcal{C}$ have heat contribution $h_j \leq 1$, this schedule is feasible. On the other hand, our algorithm schedules in every processor a job of $\mathcal{A}$ in the first slot, a job of $\mathcal{B}$ in the slot $k+2$, and k jobs of $\mathcal{C}$ in slots $k+3, k+4, \cdots, 2k+2$. Therefore, the ratio achieved by our algorithm is $\frac{2k+2}{k+2} \simeq 2$.

6.2 Maximum and Average Temperature Minimization

Next, we consider multiprocessor problems in which temperature is the optimization goal. In these problems, there is no explicit threshold on the processors' temperatures. The lack of such a threshold is counterbalanced by studying the problems of minimizing the maximum and average temperature of a schedule, i.e. $\Theta_{max} = \max_t\{\Theta_t\}$ and $\sum_t \Theta_t$. For the problem $\mathbf{S, P|p_j = 1, d_j = d, h_j|\Theta_{max}}$ we propose a tight 4/3-approximation algorithm and we show that the problem $\mathbf{S, P|p_j = 1, d_j = d, h_j|\sum \Theta_t}$ is polynomially solvable.

In these problems, we are given a set of n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ and a set of m parallel identical processors $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$. Each job $J_j \in \mathcal{J}$ is has a unit processing time $p_j = 1$, a zero release date $r_j = 0$ and a deadline $d_j = d$. Moreover, J_j is associated with a heat contribution h_j. We partition the time into unit-length slots $[0,1), [1,2), \ldots, [t,t+1), \ldots$ etc. Consider a processor $P_i \in \mathcal{P}$. At every time slot $[t,t+1)$, either a single job is executed on P_i during the whole slot or P_i is idle. If the temperature of a processor $P_i \in \mathcal{P}$ is Θ_t at time t and the job $J_j \in \mathcal{J}$ is executed on P_i during the time slot $[t, t+1)$, then the processor's temperature at time $t+1$ becomes equal to

$$\Theta_{t+1} = \frac{\Theta_t + h_j}{2}$$

On the other hand, if P_i is idle during $[t, t+1)$, then

$$\Theta_{t+1} = \frac{\Theta_t}{2}$$

The initial temperature at time $t = 0$ is $\Theta_0 = 0$.

For any instance of the maximum or average temperature problems, any schedule of length at least $\lceil \frac{n}{m} \rceil$ is feasible, independently of the range of the jobs' heat contributions. However, the optimum value of our objectives depends on the time available to execute the given set of jobs: the maximum or average temperature of a schedule of length equal to $\lceil \frac{n}{m} \rceil$ is, clearly, greater than that of a schedule of longer length, where we are allowed to introduce idle slots. In what follows, we are interested in minimizing these two objective functions with respect to a given schedule length (makespan or deadline) of $d \geq \lceil \frac{n}{m} \rceil$. Such a schedule will contain $md - n$ idle slots and we can consider them as executing $md - n$ fictitious jobs of heat contribution equal to zero. This length d is part of our problems' instances, denotes the time available to complete the execution of all the jobs and represents the need to complete them within a given time at the price of higher temperatures. Thus, in both problems we consider (minimizing the maximum and the average temperature) we are accounting the temperatures at the end of any of the md slots available on the m processors.

Maximum Temperature Minimization

Now, we turn our attention to the problem of minimizing the maximum temperature, i.e. $\mathbf{T}, \mathbf{P}|\mathbf{p_j} = \mathbf{1}, \mathbf{d_j} = \mathbf{d}, \mathbf{h_j}|\mathbf{\Theta_{max}}$. In the sequel, we will denote by Θ^*_{max} the maximum temperature of an optimal schedule.

We start with the observation that any algorithm for this problem achieves a 2 approximation ratio. Indeed, it holds that $\Theta^*_{\max} \geq h_{max}/2$, no matter how we schedule the job of maximum heat contribution. It also holds that for any algorithm, $\Theta_{\max} \leq h_{max}$, with $\Theta_{\max}$ being the maximum temperature of the algorithm's schedule. Therefore, $\Theta_{\max} \leq 2 \cdot \Theta^*_{\max}$.

To improve this trivial ratio we propose the Algorithm 6.2 below, which is based on the intuitive idea of alternating the execution of hot and cool jobs.

Algorithm 6.2

1: Sort the jobs so that $h_1 \geq h_2 \geq \ldots \geq h_n$.
2: Using the order of Step 1, schedule the $\lceil \frac{d}{2} \rceil m$ hottest jobs to the *odd* slots of the processors using Round-Robin.
3: Using the reverse order of Step 1, schedule the $\lfloor \frac{d}{2} \rfloor m$ coolest jobs to the *even* slots of the processors using Round-Robin.

To elaborate a little more on how the algorithm works, note that processor P_1 will be assigned the job J_1, followed by J_n, then followed by J_{m+1}, and then by J_{n-m} and this alternation of hot and cool jobs will continue till the end of the schedule. Similarly processor P_2 will be assigned the jobs J_2, J_{n-1}, J_{m+2}, J_{n-m-1}, and so on. The schedule is illustrated further in Table 6.1.

To analyze the Algorithm 6.2, we start with the lemma below, which is implied by the Round-Robin scheduling of jobs in Steps 2 and 3 of the algorithm.

Lemma 6.3. *In the schedule returned by Algorithm 6.2:*
(i) A job J_j, $j \geq (\lfloor \frac{d}{2} \rfloor + 1)m + 1$, is succeeded by the job $J_{n-j+m+1}$.
(ii) A job J_j, $m + 1 \leq j \leq \lceil \frac{d}{2} \rceil m$, is preceded by the job $J_{n-j+m+1}$.

P_1	J_1	J_n	J_{m+1}	J_{n-m}	J_{2m+1}	...
P_2	J_2	J_{n-1}	J_{m+2}	J_{n-m-1}	J_{2m+2}	...
...	...	...	...	...	...	...
P_m	J_m	J_{n-m+1}	J_{2m}	J_{n-2m+1}	J_{3m}	...

Table 6.1: The schedule produced by Algorithm 6.2.

The maximum temperature may occur at various points of the schedule of Algorithm 6.2. The next lemma states that one of these points satisfies a certain property regarding the heat contribution of the job executed right before.

Lemma 6.4. *In the schedule returned by Algorithm 6.2, the maximum temperature is achieved after the execution of a job J_j, with $j \leq (\lfloor \frac{d}{2} \rfloor + 1)m$.*

Proof. Assume that all the points where the maximum temperature $\Theta_{\max}$ occurs are after the execution of a job J_j, with $j \geq (\lfloor \frac{d}{2} \rfloor + 1)m + 1$. By Lemma 6.3, such a job is succeeded by a job $J_{j'}$, $j' = n - i + m + 1$, in the schedule returned by Algorithm 6.2. It is easy to check that $j > j'$, hence $h_{j'} \geq h_j$. Let $\Theta, \Theta' \leq \Theta_{\max}$ be the temperatures before the execution of J_j and after the execution of $J_{j'}$, respectively. Then, $\Theta_{\max} = \frac{\Theta + h_j}{2}$ and $h_j \geq \Theta_{\max}$, since $\Theta_{\max} \geq \Theta$. Moreover, $\Theta' = \frac{\Theta_{\max} + h_{j'}}{2} \geq \Theta_{\max}$, since $h_{j'} \geq h_j$. This implies that $\Theta' = \Theta_{\max}$, since $\Theta' \leq \Theta_{\max}$. But this means that the maximum temperature is also achieved after the execution of job $J_{j'}$, which is a contradiction because

$$j' = n - j + m + 1 \leq m(d - \lfloor \tfrac{d}{2} \rfloor) \leq m(\lfloor \tfrac{d}{2} \rfloor + 1)$$

contrary to what we assumed in the beginning of the proof. □

Lemma 6.5. *For the maximum temperature of an optimal schedule it holds that $\Theta^*_{max} \geq \frac{h_{n-j+m+1}}{4} + \frac{h_j}{2}$, for any $j \geq m + 1$.*

Proof. Consider a job J_j and let $J_{j'}$ be its previous job in the same processor in an optimal schedule $\mathcal{S}^*$. The jobs executed in the first slot of each processor in $\mathcal{S}^*$ do not have a previous one. To simplify the presentation of our proof, we assume that they are preceded by hypothetical jobs $J_{n+j''}$, $1 \leq j'' \leq m$.

If $j' \leq n - j + m + 1$, then $\Theta^*_{max} \geq \frac{h_{j'}}{4} + \frac{h_j}{2} \geq \frac{h_{n-j+m+1}}{4} + \frac{h_j}{2}$, since $h_{j'} \geq h_{n-j+m+1}$.

If $j' > n - j + m + 1$, then let $\mathcal{B} = \{J_{n-j+m+2}, J_{n-j+m+3}, \ldots, J_n, J_{n+1}, \ldots, J_{n+m}\}$ and let $\mathcal{A}$ be the set of jobs that precede the jobs $J_1, J_2, \ldots, J_{j-1}$ in the optimal schedule. Clearly, $|\mathcal{B}| = |\mathcal{A}| = j - 1$, $J_{j'} \in \mathcal{B}$ and $J_{j'} \notin \mathcal{A}$ since $J_{j'}$ precedes J_j in $\mathcal{S}^*$.

Therefore, there is a job $J_{k'} \in \mathcal{A}$ such that $J_{k'} \notin \mathcal{B}$, that is $k' < n - j + m + 2$. The job $J_{k'}$ precedes a job J_k in $\mathcal{S}^*$ and since $J_{k'} \in \mathcal{A}$ it follows, by the definition of the set $\mathcal{A}$, that $k < j$. Hence, $\Theta^*_{max} \geq \frac{h_{k'}}{4} + \frac{h_k}{2} \geq \frac{h_{n-j+m+1}}{4} + \frac{h_j}{2}$, since $h_k \geq h_j$ and $h_{k'} \geq h_{n-j+m+1}$. □

Theorem 6.4. *Algorithm 6.2 achieves a $\frac{4}{3}$ approximation ratio for* $\mathbf{T}, \mathbf{P}|\mathbf{p_j} = \mathbf{1}, \mathbf{d_j} = \mathbf{d}, \mathbf{h_j}|\ \mathbf{\Theta_{max}}$.

Proof. By Lemma 6.4 the maximum temperature in the schedule, $\mathcal{S}$, obtained by Algorithm 6.2 occurs after the execution of a job J_j, $j \leq (\lfloor \frac{d}{2} \rfloor + 1)m$ (the maximum may be achieved in other timeslots as well).

If $1 \leq j \leq m$, then the maximum occurs at the first processor and $\Theta_{\max} = \frac{h_1}{2} \leq \Theta^*_{\max}$ and, hence, the algorithm returns an optimal schedule.

If $m+1 \leq j \leq \lceil \frac{d}{2} \rceil m$ then by Lemma 6.3, the job J_j is preceded in the schedule $\mathcal{S}$ by the job $J_{n-j+m+1}$. Let Θ be the temperature before the execution of the job $J_{n-j+m+1}$. By Lemma 6.5, and since $\Theta \leq \Theta_{\max}$, $\Theta_{\max} = \frac{\Theta}{4} + \frac{h_{n-j+m+1}}{4} + \frac{h_j}{2} \leq \frac{\Theta_{\max}}{4} + \Theta^*_{\max}$. Hence, $\Theta_{\max} \leq \frac{4}{3} \cdot \Theta^*_{\max}$.

Note that if d is odd, then $\lceil \frac{d}{2} \rceil m = (\lfloor \frac{d}{2} \rfloor + 1)m$ and the analysis of the previous case holds. Hence the only remaining case is that d is even and $\lceil \frac{d}{2} \rceil m + 1 \leq j \leq (\lfloor \frac{d}{2} \rfloor + 1)m$. For this case, let $\Theta' \leq \Theta_{\max}$ be the temperature before the execution of J_j. Then, $h_j \geq \Theta_{\max}$, since $\Theta_{\max} = \frac{\Theta' + h_j}{2}$ and $\Theta_{\max} \geq \Theta'$. Thus, there are at least $\lceil \frac{d}{2} \rceil m + 1$ jobs of heat contribution at least $\Theta_{\max}$. Note that, in any schedule, each processor can execute at most $\lceil \frac{d}{2} \rceil$ jobs without any pair of them scheduled in two consecutive slots. Hence, in an optimal schedule, there are at least two jobs J_k and J_ℓ, $k, \ell \leq j$, of heat contribution at least $\Theta_{\max}$ executed in consecutive slots in the same processor. Therefore, $\Theta^*_{\max} \geq \frac{h_k}{4} + \frac{h_\ell}{2} \geq \frac{\Theta_{\max}}{4} + \frac{\Theta_{\max}}{2} = \frac{3}{4} \cdot \Theta_{\max}$, that is $\Theta_{\max} \leq \frac{4}{3} \cdot \Theta^*_{\max}$. □

For the tightness of the analysis of Algorithm 6.2 consider an instance of m processors, mn^2 jobs and $d = n^2$; suppose that there are mn hot jobs of heat contribution $h = 2$ and $mn(n-1)$ cool jobs of heat contribution $h = \epsilon$. We consider n to be sufficiently large and that ϵ tends to 0. The algorithm in each processor alternates n hot jobs with $n-1$ cool jobs and schedules $n(n-2)+1$ cool jobs at the end. The maximum temperature of the algorithm's schedule is attained exactly after the execution of the last hot job on each processor. This job is executed at slot $2n-1$, and thus $\Theta_{max} = \frac{2}{2^{2n-1}} + \frac{\epsilon}{2^{2n-2}} + \frac{2}{2^{2n-3}} + \frac{\epsilon}{2^{2n-4}} + \ldots + \frac{\epsilon}{2^2} + \frac{2}{2^1} \simeq 2\frac{\frac{1}{2}}{1-\frac{1}{4}} = \frac{4}{3}$. On the other hand, the optimal solution alternates in each processor a hot job with $n-1$ cool jobs. The temperature before the execution of any hot job tends to zero and the maximum temperature is one.

Average Temperature Minimization

Subsequently, we look at the problem of minimizing the average temperature, that is $\mathbf{T}, \mathbf{P}|\mathbf{p_j = 1}, \mathbf{d_j = d}, \mathbf{h_j}|\sum \mathbf{\Theta_t}$, instead of the maximum temperature. We will again consider a schedule length d and assume that the number of jobs is $n = md$. Contrary to the maximum temperature, we show that minimizing the average temperature of a schedule is solvable in polynomial time. Our algorithm is based on the following lemma.

Lemma 6.6. *In any optimal solution for the average temperature, jobs are scheduled in a coolest first order, i.e., for any pair of jobs $J_j, J_{j'}$ such that $h_j > h_{j'}$ scheduled at slots t and t', respectively, it holds that $t' \leq t$, regardless of the processor they are assigned to.*

Proof. Consider the job J_j to be scheduled at slot t of some processor P_i in a schedule $\mathcal{S}$. The contribution of job J_j to the temperature of the s-th slot of processor P_i (with $t \leq s \leq d$), is $\frac{h_j}{2^{s-t+1}}$, while this job does not affect the temperature of any other slot in any processor. Hence, the contribution of job J_j to the objective function, $\sum \Theta_t$, of schedule $\mathcal{S}$ is

$$\sum_{s=t}^{d} \frac{h_j}{2^{s-t+1}} = h_j \cdot \sum_{s=1}^{d-t+1} \frac{1}{2^s} = h_j \cdot (1 - \frac{1}{2^{d-t+1}}) = h_j \cdot \frac{2^{d+1} - 2^t}{2^{d+1}}.$$

Therefore, the later job J_j is scheduled, the smaller its contribution to the objective function becomes.

Assume, now, that in an optimal schedule $\mathcal{S}^*$ the job J_j is scheduled at slot t of some processor, while the job $J_{j'}$ at slot $t' > t$ in any processor. By swapping the execution of this pair of jobs the contribution of the job J_j to the objective function decreases by $h_j \cdot \frac{2^{t'}-2^t}{2^{d+1}}$ and the contribution of job $J_{j'}$ increases by $h_{j'} \cdot \frac{2^{t'}-2^t}{2^{d+1}}$. As $h_j > h_{j'}$, it follows that the resulting schedule contradicts the optimality of the schedule $\mathcal{S}^*$ and this completes the proof of the lemma. □

The previous lemma leads directly to the next simple algorithm.

Algorithm 6.3

1: Sort the jobs so that $h_1 \leq h_2 \leq \ldots \leq h_n$.
2: According to this order schedule the jobs to processors using Round-Robin.

Algorithm 6.3 finds a schedule in $O(n \log n)$ time. The optimality of this schedule follows directly by the Round-Robin scheduling of the jobs in non-decreasing order of their heat contributions and Lemma 6.6.

Theorem 6.5. *An optimal schedule for the problem* $\mathbf{T, P|p_j = 1, d_j = d, h_j|\sum \Theta_t}$ *of minimizing the average temperature can be found in polynomial time.*

In what follows, we consider a time-dependent weighted version of average temperature minimization. In particular, we consider each slot t of every processor P_i to be associated with a given positive weight $w_{i,t}$, $1 \leq t \leq d$, and our problem is denoted as $\mathbf{T, P|p_j = 1, d_j = d, h_j|\sum w_{i,t}\Theta_{i,t}}$. The weights $w_{i,t}$ could represent the interest of the system manager to keep its processors/computers cool during specific time periods of peak loads. This leads to some special, but more practical cases, of our formulation where the weights of some slots (e.g., the slot corresponding to some given time t in all processors, or an interval of consecutive slots for some processor) could be considered equal. Moreover, our analysis allows the weight of the t-th slot of processor P_i to depend on the processor too and we denote this by $w_{i,t}$, $1 \leq t \leq d$, $1 \leq i \leq m$.

Similarly with the unweighted case, we consider a job J_j of heat contribution h_j scheduled in the t-th slot of processor P_i in a schedule $\mathcal{S}$. The contribution of this job to the weighted temperature of the s-th slot of processor P_i, with $t \leq s \leq d$, is $w_{i,s} \cdot \frac{h_j}{2^{s-t+1}}$, and this job does not affect the temperature of any other slot in any processor. Hence, the contribution of job J_j to the total weighted temperature of the schedule $\mathcal{S}$ is $\sum_{s=t}^{d} w_{i,s} \cdot \frac{h_j}{2^{s-t+1}} = h_j \cdot \sum_{s=t}^{d} \frac{w_{i,s}}{2^{s-t+1}}$. Clearly, the quantity $c_{i,t} = \sum_{s=t}^{d} \frac{w_{i,s}}{2^{s-t+1}}$ is a constant that depends only on the slot t of processor P_i and not on the job executed in this slot.

Based on this, we transform our problem to a weighted bipartite matching problem and we prove the next theorem.

Theorem 6.6. *The problem* $\mathbf{T, P|p_j = 1, d_j = d, h_i|\sum w_{i,t}\Theta_{i,t}}$ *of minimizing the weighted average temperature is polynomially solvable.*

Proof. We transform the problem to a weighted bipartite matching problem. Consider a complete bipartite graph $G = (V, U; E)$ where the vertices in V correspond to the n jobs and the vertices in U to the $m \cdot d$ slots available in all processors. We set the weight of the edge between a job J_j and the slot t of processor P_i to be equal to $h_j \cdot c_{i,t}$. Hence, the weight of this edge represents the contribution of job J_j to the objective function, if it is scheduled in slot t of processor P_i. A perfect matching in the graph G corresponds to a feasible schedule and the weight of such a matching to the value of the objective function for this schedule. Therefore, a minimum weight perfect matching corresponds to an optimal solution for our problem. Such a matching can be found in polynomial time. □

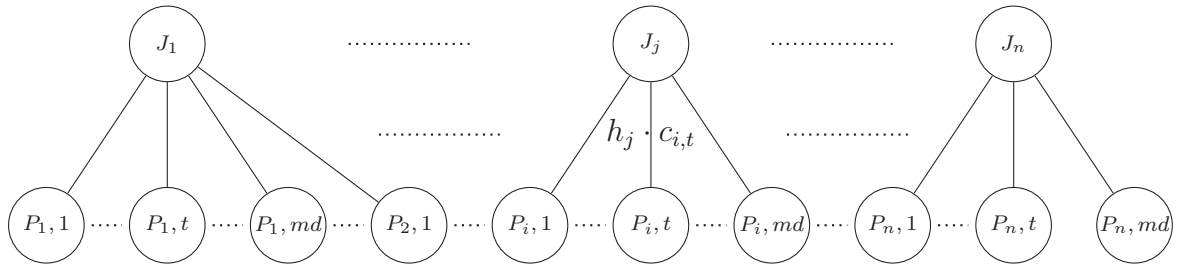


Figure 6.2: The bipartite graph for $\mathbf{T}, \mathbf{P}|\mathbf{p_j} = \mathbf{1}, \mathbf{d_j} = \mathbf{d}, \mathbf{h_i}| \sum \mathbf{w_{i,t}\Theta_{i,t}}$.

Chapter 7

Conclusion

In this thesis, we considered energy and temperature aware scheduling problems on various computing environments with different optimization goals.

Initially, we studied non-preemptive speed scaling problems with the objective of minimizing the energy. In order to solve such problems, we applied the intuitive idea of transforming optimal preemptive schedules to non-preemptive ones. We showed that this approach does not lead to constant-factor approximation algorithms for arbitrary instances. However, we obtained a 2^{α}-approximation algorithm for the single processor problem $\mathbf{S}, \mathbf{1}|\mathbf{w_j} = \mathbf{w}, \mathbf{r_j}, \mathbf{d_j}|\mathbf{E}$ with equal work jobs. An intriguing open question concerns the complexity status of this problem, i.e. whether it is polynomial or $\mathcal{NP}$-hard. By applying the same idea, we proposed a $(2 - \frac{1}{m})^{\alpha-1}$ approximation algorithm for the multiprocessor non-preemptive problem $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{agrbl}|\mathbf{E}$ with agreeable instances.

One way for solving an energy aware problem is by formulating it as a convex program. Note that a convex program can be solved in polynomial time with the Ellipsoid algorithm. However, we may obtain a faster algorithm for such a problem in the following way. We can first apply the well-known KKT conditions to the convex programming formulation of the problem and deduce a set of properties which are necessary and sufficient for optimality. Then, it suffices to derive an algorithm which always produces solutions satisfying these properties. Following this strategy, we proposed an optimal greedy algorithm for the problem of minimizing the maximum lateness with a budget of energy $\mathbf{S}, \mathbf{1}||\mathbf{L_{max}}(\mathbf{E})$ and an optimal algorithm for the multiprocessor migratory problem $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$ of minimizing the energy which is based on repeated maximum flow computations.

Subsequently, we observed that convex cost flow formulations fit well for solving energy minimization problems. Specifically, we showed that the problems $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$ and $\mathbf{S}, \mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|\mathbf{E}$ can be solved in polynomial time by using as a black box an optimal convex cost flow algorithm on appropriate graphs. An interesting future direction is to investigate further extensions of this idea in the speed scaling setting.

Next, we proposed another optimal algorithm for the energy minimization problem $\mathbf{S}, \mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|\mathbf{E}$ which is based on a primal-dual schema in the context of convex programming and KKT conditions. This algorithm is much faster than the convex cost flow algorithm when it holds that $n \neq m$. Nevertheless, new ideas are required in order to define a faster algorithm for the case where $n = m$. The primal-dual method seems

to be a useful tool in obtaining algorithms for speed scaling problems. For instance, we could expect an optimal primal-dual algorithm for $\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$ which would be faster than the best known algorithm for this problem.

Another technique that we used in order to tackle speed scaling problems is by solving configuration linear programs and applying randomized rounding. With this approach we obtained a near-optimal algorithm for the problem $\mathbf{S}, \mathbf{R}|\mathbf{w_{i,j}}, \mathbf{r_{i,j}}, \mathbf{d_{i,j}}, \mathbf{mgtn}|\mathbf{E}$ and a constant factor approximation algorithm for $\mathbf{S}, \mathbf{R}|\mathbf{w_{i,j}}, \mathbf{r_{i,j}}, \mathbf{d_{i,j}}, \mathbf{pmtn}|\mathbf{E}$ on heterogeneous environments. For the latter problem, our algorithm achieves the same approximation ratio with the best-known algorithm for the case where the processors are homogeneous. So, any improvement of our algorithm or an inapproximability result should address the homogeneous case first. Through a transformation of the single-processor non-preemptive problem to a multiprocessor preemptive problem combined with randomized rounding of an integer configuration linear program, we improved the best-known algorithm for $\mathbf{S}, \mathbf{1}|\mathbf{r_j}, \mathbf{d_j}|\mathbf{E}$. For $\alpha = 3$, we reduced the approximation ratio from 2048 to 20. Further improving this approximation ratio or obtaining an inapproximability result is a challenging open question.

Another important open question in speed scaling is the complexity status of the fundamental problem $\mathbf{S}, \mathbf{1}|\mathbf{r_j}, \mathbf{pmtn}|\sum \mathbf{C_j}$ of minimizing the average completion time under a budget of energy. In this thesis, we showed that the problem is polynomially solvable when the jobs have equal release dates. Moreover, an optimal polynomial time algorithm is known for the special case of the problem where the jobs have unit works.

The following table summarizes the main results of this thesis for speed scaling problems.

Technique	Problem	Result	Section
Preemptive to Non-Preemptive	$\mathbf{S}, \mathbf{1}|\mathbf{w_j} = \mathbf{w}, \mathbf{r_j}, \mathbf{d_j}|\mathbf{E}$	2^α-approx	2.2
	$\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{agrbl}|\mathbf{E}$	$(2 - \frac{1}{m})^{\alpha-1}$-approx	3.2
KKT and Greedy	$\mathbf{S}, \mathbf{1}||\mathbf{L_{max}}(\mathbf{E})$	OPT	2.3
	$\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$	OPT	3.1
Batched Algorithm	$\mathbf{S}, \mathbf{1}|\mathbf{r_j}|\mathbf{L_{max}} + \beta\mathbf{E}$	2-compet	2.3
Convex Cost Flow	$\mathbf{S}, \mathbf{P}|\mathbf{r_j}, \mathbf{d_j}, \mathbf{mgtn}|\mathbf{E}$	OPT	3.1
	$\mathbf{S}, \mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|\mathbf{E}$	OPT	5.1
Primal-Dual	$\mathbf{S}, \mathbf{O}|\mathbf{d_j} = \mathbf{d}, \mathbf{pmtn}|\mathbf{E}$	OPT	5.1
Configuration LP Randomized Rounding	$\mathbf{S}, \mathbf{R}|\mathbf{w_{i,j}}, \mathbf{r_{i,j}}, \mathbf{d_{i,j}}, \mathbf{mgtn}|\mathbf{E}$	OPT	4.1
	$\mathbf{S}, \mathbf{R}|\mathbf{w_{i,j}}, \mathbf{r_{i,j}}, \mathbf{d_{i,j}}, \mathbf{pmtn}|\mathbf{E}$	$\tilde{B}_\alpha$-approx	4.2
	$\mathbf{S}, \mathbf{J}|\mathbf{w_{i,j}}, \mathbf{r_{i,j}}, \mathbf{d_{i,j}}, \mathbf{pmtn}|\mathbf{E}$	$\tilde{B}_\alpha$-approx	5.2
	$\mathbf{S}, \mathbf{1}|\mathbf{r_j}, \mathbf{d_j}|\mathbf{E}$	$2^{\alpha-1}\tilde{B}_\alpha$-approx	2.2, 4.2

Table 7.1: Main Results of the Thesis.

Finally, we considered temperature-aware scheduling problems under the discrete thermal problem and we proposed constant factor approximation algorithms for the problems $\mathbf{T}, \mathbf{P}|\mathbf{p_j} = \mathbf{1}, \mathbf{h_j}|\mathbf{C_{max}}(\mathbf{\Theta})$ of minimizing the makespan under a temperature threshold and $\mathbf{T}, \mathbf{P}|\mathbf{p_j} = \mathbf{1}, \mathbf{d_j} = \mathbf{d}, \mathbf{h_j}|\mathbf{\Theta_{max}}$ of minimizing the maximum temperature. For the former problem we obtained a $2 + \epsilon$-approximation algorithm while for the latter one our algorithm achieves a 4/3-approximation ratio. Improving these results is an interesting

future direction even for the single processor case. Another important open question in the context of the discrete thermal model is whether we can improve the best known algorithm for the online problem of maximizing the throughput which is 2-competitive.

Appendix A

General Form of KKT Conditions

In this appendix, we give the general form of the KKT conditions. Assume that we are given the following convex program.

$$\begin{aligned} &\min f(x) && \\ &g_i(x) \leq 0 && 1 \leq i \leq m \\ &h_j(x) = 0 && 1 \leq j \leq \ell \\ &x \in \mathbf{R}^n && \end{aligned}$$

Suppose that the program is strictly feasible, i.e. there is a point x such that $g_i(x) < 0$ and $h_j(x) = 0$ for all $1 \leq i \leq m$ and $1 \leq j \leq \ell$, where all functions g_i and h_j are differentiable at x. Let λ_i and μ_j be the dual variables associated to the constraints $g_i(x) \leq 0$ and $h_j(x) = 0$, respectively. The Karush-Kuhn-Tucker (KKT) conditions are the following.

$$g_i(x) \leq 0 \qquad 1 \leq i \leq m \tag{A.1}$$

$$h_j(x) = 0 \qquad 1 \leq j \leq \ell \tag{A.2}$$

$$\lambda_i \geq 0 \qquad 1 \leq i \leq m \tag{A.3}$$

$$\lambda_i g_i(x) = 0 \qquad 1 \leq i \leq m \tag{A.4}$$

$$\nabla f(x) + \sum_{i=1}^{m} \lambda_i \nabla g_i(x) + \sum_{j=1}^{\ell} \mu_j \nabla h_j(x) = 0 \tag{A.5}$$

KKT conditions are necessary and sufficient for solutions $x \in \mathbf{R}^n$, $\lambda \in \mathbf{R}^m$ and $\mu \in \mathbf{R}^\ell$ to be primal and dual optimal, where $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_m)$ and $\mu = (\mu_1, \mu_2, \ldots, \mu_\ell)$. We refer to the conditions (A.1) and (A.2) as primal feasible, to the (A.3) as dual feasible, to the (A.4) as complementary slackness and and to the (A.5) as stationarity conditions, respectively.

Appendix B

KKT Conditions for Maximum Lateness plus Energy

Here we apply the KKT conditions to a convex programming formulation for the problem $\mathbf{S}, \mathbf{1}||\mathbf{L_{max}} + \beta\mathbf{E}$. As we showed in Section 2.3, this problem can be formulated as the following convex program.

$$\min L + \beta \sum_{j=1}^{n} w_j s_j^{\alpha-1} \tag{B.1}$$

$$C_j + q_j \leq L \qquad 1 \leq j \leq n \tag{B.2}$$

$$\frac{w_1}{s_1} \leq C_1 \tag{B.3}$$

$$C_{j-1} + \frac{w_j}{s_j} \leq C_j \qquad 2 \leq j \leq n \tag{B.4}$$

$$L, C_j, s_j \geq 0 \qquad 1 \leq j \leq n \tag{B.5}$$

By applying the KKT conditions we get the following lemma which describes some necessary and sufficient properties for a feasible schedule to be optimal.

Lemma B.1. *There is an optimal schedule for the maximum lateness plus energy problem satisfying the following properties.*
(i) Each job J_j runs at a constant speed s_j.
(ii) Jobs are scheduled according to the EDD rule.
(iii) Jobs are consecutively executed without any idle period.
(iv) The last job is critical, i.e., $L_n = L_{max}$.
(v) Every non-critical job J_j has equal speed with the job J_{j+1}, i.e. $s_j = s_{j+1}$.
(vi) Jobs are executed in non-increasing speeds, i.e., $s_j \geq s_{j+1}$.
(vii) The job executed first runs at speed $s_1 = (\frac{1}{(\alpha-1)\beta})^{\frac{1}{\alpha}}$.

Proof. The Properties (i) and (ii) can be easily verified through simple exchange arguments and have been discussed in Section 2.3. We prove the remaining properties by applying the KKT conditions to the above convex program.

To the constraints (B.2), (B.3) and (B.4), we associate the dual variables λ_j, μ_1, μ_j, respectively. Without loss of generality, we may assume that $L, C_j, s_j > 0$, for $1 \leq j \leq n$,

in any optimal schedule. Hence, by complementary slackness conditions, we get that the dual variables associated to the constraints (B.5) are equal to zero.

Stationarity conditions give that

$$\begin{aligned}
\nabla(L+\beta\sum_{j=1}^{n} w_j s_j^{\alpha-1}) + \sum_{j=1}^{n}\mu_j\nabla(C_j+q_j-L) \quad &+ \\
\mu_1\nabla(\frac{w_1}{s_1}-C_1) + \sum_{j=2}^{n}\mu_j\nabla(C_{j-1}+\frac{w_j}{s_j}-C_j) \quad &= \quad 0 \;\Rightarrow \\
(1-\sum_{j=1}^{n}\lambda_j)\nabla L + \sum_{j=1}^{n-1}(\lambda_j-\mu_j+\mu_{j+1})\nabla C_j \quad &+ \\
(\lambda_n-\mu_n)\nabla C_n + \sum_{j=1}^{n}\left((\alpha-1)\beta w_j s_j^{\alpha-2} - \lambda_j w_j s_j^{-2}\right)\nabla s_j \quad &= \quad 0
\end{aligned}$$

The above equation gives equivalently that

$$\sum_{j=1}^{n}\lambda_j = 1 \tag{B.6}$$

$$\lambda_j = \mu_j - \mu_{j+1} \quad \text{for} \quad 1 \le j \le n-1 \tag{B.7}$$

$$\lambda_n = \mu_n \tag{B.8}$$

$$\mu_j = (\alpha-1)\beta s_j^{\alpha} \tag{B.9}$$

Furthermore, complementary slackness conditions can be stated as

$$\lambda_j(C_j+q_j-L) = 0 \qquad 1 \le j \le n \tag{B.10}$$

$$\mu_1(\frac{w_1}{s_1}-C_1) = 0 \tag{B.11}$$

$$\mu_j(C_{j-1}+\frac{w_j}{s_j}-C_j) = 0 \qquad 2 \le j \le n \tag{B.12}$$

(iii) Since $s_j > 0$, (B.9) gives that $\mu_j > 0$ for each $1 \le j \le n$. Hence, by (B.11) and (B.12) we have that $C_1 = \frac{w_1}{s_1}$ and $C_j = C_{j-1} + \frac{w_j}{s_j}$ for $2 \le j \le n$. Therefore, there is no idle period in an optimal schedule.

(iv) Since $s_n > 0$, by (B.9) it follows that $\mu_n > 0$ and due to (B.8), $\lambda_n > 0$. So, the last job to finish is always a critical job by (B.10).

(v) Because of (B.10), if a job is non-critical, then $\lambda_j = 0$. Therefore, by (B.7) and (B.9) we have, respectively, that $\lambda_j = 0 \Rightarrow \mu_j = \mu_{j+1} \Rightarrow s_j = s_{j+1}$ which means that each non-critical job J_j has equal speed with the job J_{j+1}.

(vi) By dual feasibility conditions, $\lambda_j \ge 0$. Therefore, (B.7) and (B.9) give that $\mu_j \ge \mu_{j+1}$ and $s_j \ge s_{j+1}$, respectively. Thus, the jobs will be executed in non-increasing order of speeds.

(vii) By plugging (B.7) and (B.8) into (B.6) we get that $\mu_1 = 1$. Consequently, $s_1 = (\frac{1}{(\alpha-1)\beta})^{\frac{1}{\alpha}}$ by (B.9).

□

Appendix C

Flows and Matchings

Finally, we define some problems related to flows an matchings, namely the maximum flow, the convex cost flow, the maximum matching and the minimum weighted maximum (or perfect) matching problems. All these problems are polynomially solvable (see [1]).

Maximum and Convex Cost Flows

An instance of the maximum flow problem consists of a directed graph $G = (V, A)$, where V is a set of vertices (or nodes) and $A \subseteq V \times V$ is a set of arcs between the nodes. Each arc $e \in A$ is associated with a capacity $c_e \geq 0$ which is an upper bound on the amount of flow that can cross the edge (i, j). Moreover, we are given a source node $s \in V$ and a destination node $t \in V$. An (s, t)-flow $\mathcal{F}$ is a mapping $\mathcal{F} : E \to \mathbb{R}^+$ such that

$$\sum_{u:(u,v)\in E} f_{(u,v)} = \sum_{u:(v,u)\in E} f_{(v,u)}$$

for each node $v \in V \setminus \{s, t\}$. The value $|\mathcal{F}|$ of an (s, t)-flow $\mathcal{F}$ is

$$|\mathcal{F}| = \sum_{u:(s,u)\in E} f_{(s,u)}$$

In the maximum flow problem, we want to find an (s, t)-flow $\mathcal{F}$ of maximum value.

An instance of the convex cost flow problem consists of a directed graph $G = (V, A)$, where V is a set of nodes and $A \subseteq V \times V$ is a set of arcs between the nodes. As in the case of the maximum flow problem, we are given a source node $s \in V$, a destination node $t \in V$ and each arc $e \in A$ is associated with a capacity $c_e \geq 0$. Now, each arc $e \in A$ is also specified by a cost function $g_e(x) \geq 0$, where $x \geq 0$. The function $g_e(x)$ is convex with respect to x and it is the cost incurred if x units of flow pass through the arc e. Moreover, we are given an amount of flow $|\mathcal{F}|$. The objective is to find an (s, t)-flow $\mathcal{F}$ of value $|\mathcal{F}|$ with minimum cost such that the amount of flow that crosses each edge e does not exceed the capacity c_e, for each $e \in A$.

Matchings

Assume that we are given a bipartite graph $G = (V, U; E)$, where each edge $e \in E$ has one endpoint in the set V and the one endpoint in the set U. A matching M of G is a

subset of edges, i.e. $M \subseteq E$, such that no two edges in M have a common endpoint. A matching of maximum cardinality is a matching that contains the maximum number of edges among all the possible matchings in G. A matching is perfect if it has cardinality $|V| = |U|$. In the minimum weighted maximum (or perfect) matching problem, each edge e of G is associated with a weight $w_e \geq$ and the objective is to find a maximum (perfect) matching of minimum weight.

Bibliography

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows*. Prentice Hall, 1993.

[2] Susanne Albers. Energy efficient algorithms. *Communications of the ACM*, 53(5):86–96, 2010.

[3] Susanne Albers and Antonios Antoniadis. Race to idle: New algorithms for speed scaling with a sleep state. In *Symposium on Discrete Algorithms (SODA)*, pages 1266–1285. ACM-SIAM, 2012.

[4] Susanne Albers, Antonios Antoniadis, and Gero Greiner. On multi-processor speed scaling with migration. In *Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 279–288. ACM, 2011.

[5] Susanne Albers and Hiroshi Fujiwara. Energy efficient algorithms for flow time minimization. *ACM Transactions on Algorithms*, 3(4):49, 2007.

[6] Susanne Albers, Fabian Muller, and Swen Schmelzer. Speed scaling on parallel processors. In *Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 289–298. ACM, 2007.

[7] Lachlan L. H. Andrew, Adam Wierman, and Ao Tang. Optimal speed scaling under arbitrary power functions. *SIGMETRICS Performance Evaluation Review*, 37(2):39–41, 2009.

[8] Eric Angel, Evripidis Bampis, and Vincent Chau. Low complexity scheduling algorithm minimizing the energy for tasks with agreeable deadlines. In *Latin American Theoretical Informatics Symposium (LATIN)*, pages 13–24. Springer, 2012.

[9] Antonios Antoniadis and Chien-Chung Huang. Non-preemptive speed scaling. *Journal of Scheduling*, 16(4):385–394, 2013.

[10] Leon Atkins, Guillaume Aupy, Daniel Cole, and Kirk Pruhs. Speed scaling to manage temperature. In *International Conference on Theory and Practice of Algorithms in (Computer) Systems (TAPAS)*, pages 9–20. Springer, 2011.

[11] Evripidis Bampis, Cristoph Dürr, Fadi Kacem, and Ioannis Milis. Speed scaling with power down scheduling for agreeable deadlines. *Sustainable Computing: Informatics and Systems*, 2(4):184–189, 2012.

[12] Nikhil Bansal, David P. Bunde, Ho-Leung Chan, and Kirk Pruhs. Average rate speed scaling. *Algorithmica*, 60(4):877–889, 2011.

[13] Nikhil Bansal and Ho-Leung Chan. Weighted flow time does not admit o(1)-competitive algorithms. In *Symposium on Discrete Algorithms (SODA)*, pages 1238–1244. ACM-SIAM, 2009.

[14] Nikhil Bansal, Ho-Leung Chan, Dmitriy Katz, and Kirk Pruhs. Improved bounds for speed scaling in devices obeying the cube-root rule. *Theory of Computing*, 8(9):209–229, 2012.

[15] Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. *ACM Transactions on Algorithms*, 9(2):18, 2013.

[16] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM*, 3(4):3, 2007.

[17] Nikhil Bansal, Kirk Pruhs, and Clifford Stein. Speed scaling for weighted flow time. *SIAM Journal on Computing*, 39(4):1294–1308, 2009.

[18] Philippe Baptiste. Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. In *Symposium on Discrete Algorithms (SODA)*, pages 364–367. ACM-SIAM, 2006.

[19] Philippe Baptiste, Jacques Carlier, Alexander Kononov, Maurice Queyranne, Sergey Sevastyanov, and Maxim Sviridenko. Properties of optimal schedules in preemptive shop scheduling. *Discrete Applied Mathematics*, 159(5):272–280, 2011.

[20] Philippe Baptiste, Marek Chrobak, and Christoph Dürr. Polynomial-time algorithms for minimum energy scheduling. *ACM Transactions on Algorithms*, 8(3):26, 2012.

[21] Paul C. Bell and Prudence W. H. Wong. Multiprocessor speed scaling for jobs with arbitrary sizes and deadlines. In *International Conference on Theory and Applications of Models of Computation (TAMC)*, pages 27–36. Springer, 2011.

[22] Daniel Berend and Tamir Tassa. Improved bounds on bell numbers and on moments of sums of random variables. *Probability and Mathematical Statistics*, 30.2:185–205, 2010.

[23] Brad D. Bingham and Mark R. Greenstreet. Energy optimal scheduling on multiprocessors with migration. In *International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pages 153–161. IEEE, 2008.

[24] Martin Birks, Daniel Cole, Stanley P. Y. Fung, and Huichao Xue. Online algorithms for maximizing weighted throughput of unit jobs with temperature constraints. In *Joint Conference of International Frontiers of Algorithmics Workshop and International Conference on Algorithmic Aspects of Information and Management (FAW-AAIM)*, pages 319–329. Springer, 2011.

[25] Martin Birks and Stanley P. Y. Fung. Temperature aware online scheduling with a low cooling factor. In *International Conference on Theory and Applications of Models of Computation (TAMC)*, pages 105–116. Springer, 2010.

[26] Martin Birks and Stanley P. Y. Fung. Temperature aware online algorithms for scheduling equal length jobs. In *Joint Conference of International Frontiers of Algorithmics Workshop and International Conference on Algorithmic Aspects of Information and Management (FAW-AAIM)*, pages 330–342. Springer, 2011.

[27] Martin Birks and Stanley P. Y. Fung. Temperature aware online algorithms for minimizing flow time. In *International Conference on Theory and Applications of Models of Computation (TAMC)*, pages 20–31. Springer, 2013.

[28] David P. Bunde. Power-aware scheduling for makespan and flow. *Journal of Scheduling*, 12(5):489–500, 2009.

[29] Jian-Jia Chen, Heng-Ruey Hsu, Kai-Hsiang Chuang, Chia-Lin Yang, Ai-Chun Pang, and Tei-Wei Kuo. Multiprocessor energy efficient scheduling with task migration considerations. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 101–108. IEEE, 2004.

[30] Marek Chrobak, Christoph Dürr, Mathilde Hurand, and Julien Robert. Algorithms for temperature-aware task scheduling in microprocessor systems. *Sustainable Computing: Informatics and Systems*, 1(3):241–247, 2011.

[31] Marek Chrobak, Uriel Feige, Mohammad Taghi Hajiaghayi, Sanjeev Khanna, Fei Li, and Seffi Naor. A greedy approximation algorithm for minimum-gap scheduling. In *International Conference on Algorithms and Complexity (CIAC)*, pages 97–109. Springer, 2013.

[32] Erik D. Demaine, Mohammad Ghodsi, Mohammad Taghi Hajiaghayi, Amin S. Sayedi-Roshkhar, and Morteza Zadimoghaddam. Scheduling to minimize gaps and power consumption. In *Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 46–54. ACM, 2007.

[33] Nikhil R. Devanur, Christos H. Papadimitriou, Amin Saberi, and Vijay V. Vazirani. Market equilibrium via a primal-dual algorithm for a convex program. *Journal of the ACM*, 55(5):22, 2008.

[34] Christoph Dürr, Ioannis Milis, Julien Robert, and Georgios Zois. Approximating the throughput by coolest first scheduling. In *Workshop on Approximation and Online Algorithms (WAOA)*, pages 187–200. Springer, 2012.

[35] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[36] Teofilo F. Gonzalez. A note on open shop preemptive schedules. *IEEE Transactions on Computers*, 28(10):782–786, 1979.

[37] Ronald Graham, Eugene Lawler, Jan Karel Lenstra, and Alexander Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[38] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, 1969.

[39] Gero Greiner, Tim Nonner, and Alexander Souza. The bell is ringing in speed scaled multiprocessor scheduling. In *Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 11–18. ACM, 2009.

[40] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1993.

[41] Anupam Gupta, Sungjin Im, Ravishankar Krishnaswamy, Benjamin Moseley, and Kirk Pruhs. Scheduling heterogeneous processors isn't as easy as you think. In *Symposium on Discrete Algorithms (SODA)*, pages 1242–1253. ACM-SIAM, 2012.

[42] Anupam Gupta, Ravishankar Krishnaswamy, and Kirk Pruhs. Scalably scheduling power-heterogeneous processors. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 312–323, part 1. Springer, 2010.

[43] Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the ACM*, 34(1):144–162, 1987.

[44] Wassily Hoeffding. On the distribution of the number of successes in independent trials. *Annals of Mathematical Statistics*, 27(3):713–721, 1956.

[45] Sandy Irani, Rajesh K. Gupta, and Sandeep K. Shukla. Competitive analysis of dynamic power management strategies for systems with multiple power savings states. In *Conference on Design, Automation and Test in Europe (DATE)*, pages 117–123. IEEE, 2002.

[46] Sandy Irani and Kirk Pruhs. Algorithmic problems in power management. *ACM SIGACT News*, 36(2):63–76, 2005.

[47] Sandy Irani, Sandeep K. Shukla, and Rajesh Gupta. Algorithms for power savings. *ACM Transactions on Algorithms*, 3(4):41, 2007.

[48] Tak Wah Lam, Lap-Kei Lee, Isaac Kar-Keung To, and Prudence W. H. Wong. Nonmigratory multiprocessor scheduling for response time and energy. *IEEE Transactions on Parallel and Distributed Systems*, 19(11):1527–1539, 2008.

[49] Tak Wah Lam, Lap-Kei Lee, Isaac Kar-Keung To, and Prudence W. H. Wong. Online speed scaling based on active job count to minimize flow plus energy. *Algorithmica*, 65(3):605–633, 2013.

[50] Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. *Journal of Computer and System Sciences*, 73(6):875–891, 2007.

[51] Minming Li, Becky Jie Liu, and Frances F. Yao. Min-energy voltage allocation for tree-structured tasks. *Journal of Combinatorial Optimization*, 11(3):305–319, 2006.

[52] Minming Li, Andrew C. Yao, and Frances F. Yao. Discrete and continuous min-energy schedules for variable voltage processors. *Proceedings of the National Academy of Sciences of the United States of America*, 103(11):3983–3987, 2006.

[53] Robert McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6(1):1–12, 1959.

[54] Nicole Megow and José Verschae. Scheduling on a machine with varying speed: Minimizing cost and energy via dual schedules. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 745–756. Springer, 2013.

[55] Yuri Nesterov and Arkadi Nemirovski. *Interior Point Polynomial Algorithms in Convex Programming*. SIAM, 1994.

[56] Kirk Pruhs, Patchrawat Uthaisombut, and Gerhard J. Woeginger. Getting the best response for your erg. *ACM Transaction on Algorithms*, 4(3):38, 2008.

[57] Kirk Pruhs, Rob van Stee, and Patchrawat Uthaisombut. Speed scaling of tasks with precedence constraints. *Theory of Computing Systems*, 43(1):67–80, 2008.

[58] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003.

[59] David B. Shmoys, Joel Wein, and David P. Williamson. Scheduling parallel machines on-line. *SIAM Journal on Computing*, 24(6):1313–1331, 1995.

[60] Oscar C. Vásquez. Energy in computing systems with speed scaling: Optimization and mechanisms design. In *arXiv:1212.6375*, 2012.

[61] László A. Végh. Strongly polynomial algorithm for a class of minimum-cost flow problems with separable convex objectives. In *Symposium on Theory of Computing (STOC)*, pages 27–40. ACM, 2012.

[62] F. Frances Yao, Alan J. Demers, and Scott Shenker. A scheduling model for reduced cpu energy. In *Symposium on Foundations of Computer Science (FOCS)*, pages 374–382. IEEE, 1995.