Energy Minimization, Data Movement and Uncertainty : Models and Algorithms

Konstantinos Dogeas

Thèse de doctorat en informatique
de Sorbonne Université

École Doctorale Informatique, Télécommunications et Électronique

---

Energy Minimization, Data Movement and Uncertainty: models and algorithms

---

par
Konstantinos Dogeas

présentée et soutenue publiquement
pour obtenir le grade de
DOCTEUR de SORBONNE UNIVERSITÉ

Date provisoire de soutenance: April 6, 2022

Jury

Dr. Evripidis Bampis
Sorbonne Université
Directeur de thèse – Professeur des universités

Dr. Thomas Erlebach
Durham University
Rapporteur – Professeur des universités

Dr. Dimitris Fotakis
National and Technical University of Athens
Rapporteur – Professeur des universités

Dr. Giorgio Lucarelli
Université de Lorraine
Co-encadrant – Maître de conférences

Dr. Fanny Pascual
Sorbonne Université
Co-encadrante – Maîtresse de conférences

Dr. Pierre Sens
Sorbonne Université
Examinateur – Professeur des universités

Dr. Denis Trystram
Université de Grenoble-Alpes
Examinateur – Professeur des universités

Abstract

High performance computers (HPCs) is the go-to solution for running computationally demanding applications. As the limit of energy consumption is already achieved, the need for more energy efficient algorithms is critical. Taking advantage of the core characteristics of an HPC, such as its network topology and the heterogeneity of the machines, could lead to better scheduling algorithms. In addition, designing more realistic models, that grasp the features of real-life applications, is a work in the same direction of achieving better performance. Allowing scheduling algorithms to decide either the amount of resources allocated to an application or the running speed of the resources can pave the path to new platform-aware implementations.

In the first part of the thesis, we introduce a model which takes into account both the topology and the heterogeneity of a platform by introducing two kind of machines. We augment the scheduling problem with constraints whose purpose is to implicitly reduce data movement either during parallel execution or during the communication with the file system. We propose algorithms that can decide the number of resources allocated to an application taking into consideration the extra constraints.

In the second part of the thesis, we deal with the uncertainty on part of the input and more specifically, the workload of an application, that is strictly related to the time needed for its completion. Most works in the literature consider this value known in advance. However, this is rarely the case in real-life systems. In our approach, the given workload is a worst case scenario for the execution of an application. We introduce application-specific tests that may decrease the workload of a task. Since the test (e.g. compression) takes some time, and since the amount of reduction (e.g. in size) is unknown before the completion of the test, the decision of running the test for a task or not has to be taken. We propose competitive algorithms for the problem of scheduling such tasks, in order to minimize the energy consumed in a set of speed-adjustable machines.

In the third part of the thesis, we focus on a similar setting of uncertain input and we consider a model where the processing times are not known in advance.

Here, we augment the input of the problem by introducing predicted values in place of the unknown processing times. We design algorithms that perform optimally when the predictions are accurate while remaining competitive to the best known ones otherwise.

Résumé

Les plateformes de calcul haute performance (HPC) sont la solution idéale pour exécuter des applications exigeantes en termes de calcul. Étant donné leur consommation importante en énergie, le besoin d'algorithmes plus efficaces en termes d'énergie est indispensable. De meilleurs algorithmes d'ordonnancement peuvent être conçus en exploitant les caractéristiques essentielles d'une plateforme HPC, telles que sa topologie de réseau et l'hétérogénéité de ses machines. On peut également obtenir de meilleures performances en concevant des modèles plus réalistes, qui saisissent les fonctionnalités d'applications réelles. Ainsi, permettre aux algorithmes d'ordonnancement de décider de la quantité de ressources allouées à une application, ou de la vitesse d'exécution des machines, peut ouvrir la voie à de nouvelles implémentations compatibles avec la plateforme.

Dans la première partie de la thèse, nous introduisons un modèle qui prend en compte à la fois la topologie et l'hétérogénéité d'une plateforme en introduisant deux types de machines. Nous augmentons le problème d'ordonnancement avec des contraintes dont le but est de réduire implicitement le mouvement des données pendant l'exécution des tâches sur des machines parallèles, et lors de la communication avec le système de fichiers. Nous proposons des algorithmes qui ordonnancent les tâches au cours du temps, et décident du nombre de ressources allouées à une tâche, en tenant compte de ces contraintes supplémentaires.

Dans la deuxième partie de la thèse, on s'intéresse à l'incertitude liée à la charge de travail d'une application, cette charge étant directement liée au temps nécessaire à son exécution. La plupart des travaux de la littérature considèrent cette valeur connue à l'avance. C'est cependant rarement le cas dans les systèmes réels. Dans notre approche, la charge de travail donnée est une charge possible mais qui peut éventuellement être réduite. On introduit alors des tests spécifiques à l'application qui peuvent réduire la charge de travail d'une tâche. étant donné que le test (par exemple, la compression) doit également être exécuté, et que la quantité de réduction (par exemple, la taille) est inconnue avant la fin du test, la décision d'exécuter ou non le test pour une tâche doit être prise. On propose des algorithmes compétitifs pour le problème d'ordonnancement de telles tâches, dans le but de minimiser l'énergie consommée par un ensemble de machines pour lesquelles on peut modifier la vitesse.

Dans la troisième partie de la thèse, nous nous intéressons à un contexte similaire d'entrées incertaines et nous considérons un modèle dans lequel les temps d'exécution des tâches ne sont pas connus à l'avance. Nous augmentons l'entrée du problème en introduisant des valeurs prédites des temps d'exécution. Nous concevons alors des

algorithmes qui ont d'excellentes performances lorsque les prédictions sont exactes, tout en restant compétitifs lorsque les prédictions se révèlent inexactes.

Acknowledgments

In the next few lines, I'd like to thank everyone who played a part (big or small) in this rather personal journey.

First of all my supervisors throughout my academic life. Evripidis Bampis in the thesis, Giorgio Lucarelli in the master's and the late Ioannis Milis in the bachelor's. Their altruistic priority was always to make me a better researcher, even though in the end they also made me a better person. In the same time, I want to thank my co-supervisor, Fanny Pascual, whose guidance was essential throughout the years and Alexander Kononov for the enriching collaboration.

Moreover, I want to express my luck to be surrounded by my office-mates, Anne-Elisabeth, Adele, Franco and (the newcomer) Francois, as well as by my lab-mates, Marvin, Gaspard, Anja and (the outsider) Arnaud, for creating a lively and stimulating working environment. Extra credits should be given to David for remaining a passionate individual and transmitting his ideas through our very pleasant discussions.

Furthermore, I would like to address my appreciation to people with a timeless importance to me. To my very close friends Gerasimos, Alexandros, Antonis and George D. for their almost stubborn existence in my life. To Federico, Abdallah, Afroditi, Dylan and Rutger (and the rest of my neighbors) for their contribution in maintaining a social cycle whilst in Paris. To George A., Nikos, Elli, Christos, Andreas, Manthos and Thodoris for their contribution in maintaining a balanced mental state whilst in Paris. Special thanks to Clara, who belongs in all three above categories and decided that dealing with me is a good idea.

In addition, I would like to thank my older brothers, Epameinondas and Ioannis, for each showing me their distinct approach in life, a blend of which I follow today. Finally, I'm grateful to my parents, Stavros and Liana, for providing everything they could and for the unconditional love and support they show in every step along this way.

Contents

1 Introduction

Scheduling is one of the most studied problems in computer science. In its typical setting, we have a set of jobs that need to be executed in a set of machines. The vast characteristics of these two ingredients have unraveled multiple variants, each one with different scientific importance. This diversity can grasp the important features of real life systems. Scheduling algorithms can be applied to simple problems, such as whether to rent or buy skis, to complex ones, e.g. how to program the *resource and job managing system (RJMS)* of a supercomputer. On one hand, our main goal is to create models that mirror as good as possible real life systems and on the other hand to propose algorithms that solve these models. This thesis is part of the ANR Energumen (*ANR-18-CE25-0008*) project which proposes to revisit the principles of the existing RJMS toward the evolution of large-scale parallel systems.

In our days, *High Performance Computers* (HPCs) are widely used to run applications of great societal importance. Since their debut, the goal of the scientific community is to increase their efficiency. For many years, this was achieved via the hardware of the systems: either by increasing the scale of the platform, or by introducing special purpose processors and heterogeneity on the machines (nodes), or by improving the interconnection network. However, it is now that the power consumption becomes a major constraint. For example, electricity companies set upper bounds on the power for HPC systems in different time periods during the day. Hence, HPCs' designers turned their focus on the scheduling algorithms in order to increase drastically the computing performance within the same order of magnitude in energy as today. Existing HPCs consist of more than one type of nodes like computational accelerators (GPUs due to their efficiency in specific kind of operations) as well as machines dedicated to the communication with a (usually distributed) file system, i.e. Input/Output (I/O) nodes. I/O nodes have a positive effect on reducing communication cost and they can prevent the network from acting as a bottleneck to the overall performance of the platform. As the complexity of platforms increases, the need for new, more precise, platform-oriented algorithms, which take into consideration the various features of HPCs, is crucial.

Energy Energy minimization is a crucial point for utilizing HPCs at a sustained rate. We use two mechanisms to achieve so. By allowing malleability on the resources, schedulers can decide upon which quantity of computing resources to allocate to each job taking also into account all different possible parameters/states, such as communication requirements and data movements [Non13]. Here, the number of allocated resources

for a job can be adapted, aiming at the same time to reduce the data movements by appropriate allocations, with the global goal of reducing the energy consumption.

The second mechanism to reduce energy is *speed scaling* [YDS95]. Here, the clock frequency of the processors can be adapted (also known as *Dynamic Voltage and Frequency Scaling - DVFS*) to the energy requirements. Specifically, higher speed corresponds to better performance, but higher energy consumption. To quantify this, we assume that, if a machine runs at speed $s(t)$ at a time instant t, then the power needed is $P(s(t))$. In integrated systems produced by the standard CMOS technology, the power can theoretically be described as $P(s(t)) = s(t)^3$, but in practice this exponent varies for different architectures. We consider the more general case where the power is described by the function $P(s(t)) = s(t)^\alpha$, where $\alpha > 1$ is constant. Then, the energy consumption is computed as $E = \int P(s(t))dt$.

Data Movement Energy savings can be also obtained as a consequence of data movement reductions [Non13]. This communication can be divided into intra-job (corresponding mainly to memory management and heterogeneity between the allocated nodes) and between jobs (network, interconnect design, I/O). We note that, the communication costs are taken into account implicitly: smart allocations that reduce both kind of communications are imposed to the scheduler, without measuring explicitly the congestion. Ideally, we would like to design algorithms that take into account the structure of the network to reduce data movement. We augment the scheduling problem with constraints, that do not allow interactions between any two applications, resulting in less congestion in the underlying network, and at the same time do not have a bad effect on the overall performance of the system. We propose generic scheduling algorithms for HPC platforms to minimize the makespan taking into account communication issues as well as the existence of I/O nodes.

Uncertainty The majority of the literature shares a common assumption; the workload (and hence the processing time) of each job is known either in advance or when the job becomes available (clairvoyance). In order to remedy this problem, we consider two settings, which introduce *uncertainty* on the workload. In this part of the thesis, we use simpler models with more academic interest. However, both models remain relevant to our goal for more realistic ones as they provide a first methodology to tackle uncertainty in more complex environments.

Uncertainty with Tests In the first setting, inspired by the works in [AE20; DEM+18], each job has an a priori unknown workload that can be revealed to the algorithm only after executing a query (or test) that induces a given additional job-dependent load. Alternatively, a job may be executed without any query, but in that case its workload is equal to a given upper bound. We analyze this model under the speed scaling setting and we try to understand what is the impact of the extra load induced by the queries to

the overall energy consumption of the system. This setting is motivated by the fact that a query could correspond to a code optimizer as mentioned in [DEM+18]. In that case, the code optimizer needs some extra work to process the job and potentially reduce its workload. The upper bound of a job corresponds to the work needed to take place when the code optimizer is not executed. Another possible application for this assumption is file compression. In this model, we minimize the total energy consumption following the speed scaling setting of [YDS95].

Uncertainty with Predictions In the second setting, predicted values are available in the place of the unknown processing times. There is extensive literature for the non-clairvoyant case (processing times are unknown and only the existence of a job is revealed to the algorithm), with works proposing robust algorithms that perform well without any knowledge of the processing times. However this kind of analysis is often too pessimistic. Predictions introduce a new flavor of uncertainty in the scheduling problem as one cannot know their accuracy beforehand [GGK+19; PSK18]. They present a way to bridge the gap between the clairvoyant and the non-clairvoyant setting. We say an online algorithm is *consistent*, if it performs close to the best offline algorithm, when the predictor is good. Otherwise, when the predictor is bad, the online algorithm should gracefully degrade and should perform close to the online algorithm without predictions. Our goal is to obtain algorithms that are both consistent and robust. We study this model under the objective of minimizing the sum of completion times.

1.1 Notation

Throughout this thesis, we consider a set of jobs, $\mathcal{J}$ (jobset), of cardinality n to be scheduled either on a single machine or on a set of machines of cardinality m. We call a job *malleable*[1] , if the scheduler can adapt the resources allocated to this job, either by deciding the number of computing nodes or by speed scaling. Otherwise, if a job has a fixed need in computational resources, we call it *rigid*. For a job j, we denote its finishing (completion) time by C_j. Some other basic characteristics are the following:

- **Workload (w_j) :** the amount of work needed for the completion of a job j.
- **Upper bound (u_j) :** the maximum amount of work needed for the completion of a job j.

1 Here, we follow the terminology used in [MRT07; TWY92] concerning malleable jobs, where the algorithm decides a fixed number of machines to use throughout the execution of a job. Feitelson and Rudolph in [FR96] use a more general terminology. According to their work, the algorithm can change the number of allocated machines during the execution of a malleable job. On the contrary, jobs that are similar to our case and the algorithm have to decide the number of allocated machines before the start of their execution, are called moldable.

- **Query/Test (t_j) :** the amount of work needed for a job-dependent test to complete.
- **Processing time (p_j) :** the processing time needed for the execution of job j. The processing time is a function of the workload and defined as needed for each problem. If not specified otherwise, we suppose that the time needed for a job to be executed on a single constant speed machine is equal to the job's workload, i.e. $p_j = w_j$.
- **Predicted processing time (y_j) :** the predicted time needed for the execution of a job j.
- **Release date (r_j) :** the time a job j arrives at the system, which is the earliest time at which it can start execution.
- **Deadline (d_j) :** the latest time a job j must complete its execution.

The three field expression $\alpha \mid \beta \mid \gamma$, commonly known as Graham's notation, introduced in [GLL+79], is widely used to describe scheduling problems. We present here some basic notation for the problems studied in this thesis that coincide with the literature. The first field describes the machine environment, $\alpha = \{1, P, P^{\mathrm{C}}, P^{I/O}\}$. The second one provides details on the problem-specific constraints, $\beta = \{pmtn, mgrt, r_j, online - r_j, d_j, non - clair\}$. The last one contains the objective function to be optimized (minimized), $\gamma = \{C_{\max}, \sum_j C_j, E\}$. We may extend this notation, in each chapter individually, with more problem-related constraints.

- **1 :** there is only one machine in the system.
- $\boldsymbol{P}$ **:** there are m *parallel* and *identical* machines in the system.
- $\boldsymbol{P^{\mathrm{C}}}$ **:** there are m^{C} *parallel* and *identical* machines in the system, dedicated to computations.
- $\boldsymbol{P^{I/O}}$ **:** there are $m^{I/O}$ *parallel* and *identical* machines in the system, dedicated to input/output operations.

- ***pmtn*** **:** jobs can be stopped and later resume execution.
- ***mgrt*** **:** jobs can be stopped and (later) resume execution on a different machine.
- $\boldsymbol{r_j}$ **:** the earliest time at which a job can begin execution and in addition, all release times are known in advance (offline case).
- $\boldsymbol{online - r_j}$ **:** the earliest time at which a job can begin execution and in addition, the algorithm finds out about the existence of this job at time r_j.
- $\boldsymbol{d_j}$ **:** the latest time at which a job must finish execution.

- ***non−clair*** **:** the algorithm finds out the processing time of the job only after executing it for p_j units of time.
- ***cont*** **:** a job must be executed by neighbouring machines with respect to the underlying topology (see Definition 3.1 in Chapter 3).
- ***local*** **:** a pre-specified machine must be used in the execution of a job (see Definition 3.2 in Chapter 3).

- $C_{\max}$ **:** the maximum completion time among all jobs (makespan).
- $\sum_j C_j$ **:** the total completion time.
- E **:** the total energy consumption.

1.2 Preliminaries

(In)Tractability Some problems, that we call *tractable*, can be solved optimally by a polynomial time algorithm, while other ones, called *intractable*, cannot. There is a class of problems, called $\mathcal{NP}$-complete, for which we don't know if they are tractable or not. The problems in this class have equivalent difficulty in the sense that if one of them is tractable, this would imply tractability for the rest of $\mathcal{NP}$-complete problems. Similarly, if one of them is intractable, this would imply intractability for the other problems.

Given this property, we can show that an unclassified problem, Π', belongs in the $\mathcal{NP}$-complete class using a polynomial time *reduction* from an already known $\mathcal{NP}$-complete problem Π to Π'. We claim that if there is a solution to Π, we can transform this solution, in polynomial time, to a solution for Π' and vice versa. Using this technique we prove that either the problem Π' is also $\mathcal{NP}$-complete, or we can provide a solution to any $\mathcal{NP}$-complete problem. Since it is conjectured that $\mathcal{P} \neq \mathcal{NP}$, a reduction concludes the first. To fully understand the complexity theory, we guide the reader to the book of Garey and Johnson, "Computers and Intractability" [GJ79].

Approximation Algorithm In the offline setting, where the entirety of the input is available in the beginning, we design *approximation algorithms*. Formally, consider an optimization problem and an algorithm A. For a given instance I, denote by $C_A(I)$ and $C_{OPT}(I)$ the cost of the algorithm's solution and the cost of the optimal solution, respectively. We call A a ρ-approximation algorithm for a minimization problem, if for any instance I, we have:

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

Competitive Algorithm In the online setting, where the instance is becoming available over time, we design *competitive algorithms* that need to take irrevocable decisions at each step. Here, for a given instance I and an online algorithm A, denote by $C_A(I)$ and $C_{OPT}(I)$ the cost of the online algorithm's solution and the cost of the optimal offline solution, respectively. We call A a ρ-competitive algorithm for a minimization problem, if for any instance I, we have:

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

Learning Augmented Algorithm [BMS20; LV18; PSK18] A *learning augmented algorithm*, A, receives as input a prediction P, an instance I which is revealed online, a robustness parameter λ, and outputs a solution of cost $C_A(P, I, \lambda)$. For any $0 < \lambda \leqslant 1$, we say that A is $C(\lambda)$-consistent and $R(\lambda)$-robust if the cost of the solution satisfies:

$$C_A(P, I, \lambda) \leqslant \min\{C(\lambda) \cdot S(P, I), R(\lambda) \cdot OPT(I)\}$$

where $S(P, I)$ is the cost of the output solution on input I if the algorithm follows blindly the prediction. If P is accurate ($S(P, I) \approx OPT(I)$) and we trust the prediction, we would like the performance to be close to the optimal offline. $C(\lambda)$ should approach 1 as λ approaches 0. Similarly, if there is no trust to the prediction, algorithm A should not perform much worse than the best pure online algorithm. $R(1)$ should be close to the best pure online algorithm.

1.3 Outline of the Thesis

In this thesis, we attempt to introduce models that are one step closer to real-life systems and propose efficient algorithms for these models.

In Chapter 2, we present related work that lead to the study cases considered in this thesis.

In Chapter 3, we present the first scheduling model on designing algorithms that take into consideration the topology of a machine using malleable jobs, i.e. $P^C, P^{I/O} \mid cont, local \mid C_{\max}$. We first prove that the problem with the new constraints, namely contiguity and locality, is $\mathcal{NP}$-hard. We then present approximation algorithms to solve different variants of the problem. This chapter is based on the paper with name "Scheduling Malleable Jobs under Topological Constraints" [BDK+20].

In Chapter 4, we explore the concept of uncertainty on values of the input. We introduce a speed scaling model with tests, which, if executed, can reveal a more accurate value of the processing time of a job. In Graham's notation, the problems are $1 \mid r_j, d_j \mid E$, $1 \mid online - r_j, d_j \mid E$ and $P \mid online - r_j, d_j \mid E$. We present competitive algorithms for all the variants of the problem. This chapter is based on the paper with name "Scheduling with Explorable Uncertainty" [BDK+21].

In Chapter 5, we consider a non-clairvoyant scheduling problem. We introduce a model where the processing times are unknown, yet predicted values are available. We present competitive algorithms for the problems $1 \mid online - r_j, non - clair \mid \sum_j C_j$ and $P \mid non - clair \mid \sum_j C_j$. This chapter is based on the paper with name "Scheduling with Uncertain Predictions" (under submission in IJCAI 2022).

Finally, in Chapter 6 we give the concluding remarks of the thesis as well as perspectives for future work.

2 Related Work

In this chapter, we give a not exhaustive description of existing work that is closely related to the models in this thesis. We group the results based on the minimization criterion of each model. To begin with, in Section 2.1 we consider the makespan minimization objective corresponding to the model in Chapter 3. Here, we present part of the literature about topological constraints, such as contiguity and locality, as well as works on malleable jobs. Next, in Section 2.2, we consider the energy minimization objective corresponding to the model in Chapter 4. We start by briefly describing works from the speed scaling setting, and we also present previous works in the explorable uncertainty setting. Finally, in Section 2.3, we consider the total completion time minimization objective corresponding to the model in Chapter 5. Most related past works on offline, online and non-clairvoyant models for both single and multiple machines settings are presented. In this section, we also discuss existing work on uncertain predictions.

2.1 Makespan Minimization and Topological Constraints

In this section, we first talk about scheduling rigid or malleable jobs in parallel machines in order to minimize the makespan. We then describe works that take into account the contiguity and locality constraints.

2.1.1 Parallel Machines

Rigid Jobs

The general problem for parallel machines, $P \mid\mid C_{\max}$, without constraints is *strongly-*$\mathcal{NP}$-hard. It remains $\mathcal{NP}$-hard even when there are only 2 available machines. Blazewicz et al. [BDW86] study the problem where some jobs may ask for a specific number of machines (multiprocessor jobs), i.e. $P \mid size_j \mid C_{\max}$. When the number of machines is not fixed, they show that the problem is *strongly-*$\mathcal{NP}$-hard even for unit time multiprocessor jobs. Du and Leung [DL89] show that the same problem can be solved in pseudo-polynomial time for the case of 2 or 3 available machines, and it becomes *strongly-*$\mathcal{NP}$-hard for 5 or more machines. Whether the case with 4 machines is *strongly-*$\mathcal{NP}$-hard or solvable in pseudo-polynomial time is left open. In another work, Bozoki and

Richard consider the problem where a job asks for a set of machines (not necessarily contiguous) that must all be used simultaneously for the completion of this job [BR70], i.e. $P \mid fix_j \mid C_{\max}$. They give a branch and bound type algorithm for the solution of this problem. The problem is proved to be *strongly-$\mathcal{NP}$*-hard for the case of 3 available machines by a reduction from 3-Partition [BDD+92; DST97]. If the number of machines is part of the problem, it is $\mathcal{NP}$-hard for unitary jobs, i.e. $P \mid fix_j, p_j = 1 \mid C_{\max}$. Hoogeveen et al. [HVV94] show that for this problem, there exists no polynomial time approximation algorithm with performance ratio smaller than $\frac{4}{3}$, unless $\mathcal{P} = \mathcal{NP}$. When the number of machines is fixed, Amoura et al. [ABK+02] propose polynomial time approximation schemes for both $P_m \mid fix_j \mid C_{\max}$ and $P_m \mid size_j \mid C_{\max}$. For the latter problem, an asymptotic fully polynomial time approximation scheme is proposed by Jansen in [Jan02]. In problem $P \mid set_j \mid C_{\max}$, for each job j, the set set_j describes the different subsets of simultaneously required machines that can be used for the execution of j. Note that only one choice from this set can be made. If the number of computing nodes is fixed, then a polynomial time approximation scheme for $P_m \mid set_j \mid C_{\max}$ has been presented in [CM01]. However, if m is part of the instance, there is no polynomial time approximation algorithm with ratio smaller that n^{δ}, for any $\delta > 0$, as shown in [MTC02]. A survey for multiprocessor jobs can be found in [Dro96].

Malleable Jobs[2]

The problem of scheduling malleable jobs is also shown to be *strongly-$\mathcal{NP}$*-hard by Du and Leung [DL89] in the case of non-monotonic jobs. Allocating more processors to a monotonic job decreases its execution time and increases its work. A 2-factor approximation algorithm for this version has been given by Turek et al. [TWY92]. Jansen and Porkolab [JP02] gave a PTAS for instances with a constant number of machines, while in [JT10], Jansen and Thöle proposed a PTAS when the number of machines is polynomial in the number of jobs. In the case of monotonic jobs, Mounié et al. proposed a $\frac{3}{2}$-approximation algorithm [MRT07]. More recently, Fotakis et al. studied the case of malleable job scheduling, where jobs can be executed simultaneously on multiple non-identical machines with the processing time depending on the number of allocated machines [FMP19].

2.1.2 Topological Constraints

The idea of generic topology-oriented algorithms is not new. Bladek et al. introduced the notion of *contiguous allocations* and they theoretically proved that imposing this kind of allocations does not deteriorate too much the optimal schedule [BDG+15].

2 To avoid any confusion, we remark again that we use the terminology of [MRT07; TWY92] concerning malleable jobs, where the algorithm decides a fixed number of machines to use throughout the execution of a job.

Contiguity

Scheduling rigid jobs under the contiguity constraint is closely related to the *Strip Packing* problem [KR96], the *Dynamic Storage Allocation* [BKK+04], as well as to the problem of scheduling multiprocessor jobs [ABK+02; Dro96].

Strip Packing In the strip packing problem, a set of rectangles needs to be packed in a strip of width 1 and the objective is to minimize the height. The rectangles must not overlap, cannot be rotated and they are always parallel to both x and y axis. The strip packing problem is also $\mathcal{NP}$-hard. For this problem, Steinberg shows an algorithm with absolute performance bound of 2 [Ste97]. An improvement on this result came by Harren et al. in [HJP+14] achieving a ratio of $\frac{5}{3} + \epsilon$. Different other versions of the strip packing problem have been studied. For example, Bougeret et al. give an approximation guarantee of 2 for the multiple strip packing problem [BDJ+09]. An interesting analysis was given by Han et al. in [HIY+16], where bin packing techniques are used to approximate strip packing. Moreover, pseudo-polynomial algorithms have been proposed, with the most interesting one being the one of Jansen and Rau in [JR19]. Finally, a fully polynomial time approximation scheme was given by Kenyon and Ramila in [KR00] providing a nearly optimal solution.

We relate scheduling to strip packing by considering each job as a rectangle. If related to the x-dimension, the number of parallel machines in the system defines the length of the strip. Then, the *contiguous* machine requirement of a job defines the length on the x-dimension of the rectangle while the processing time defines the length of the y-dimension. The objective of minimizing the height of the strip is translated into minimizing the makespan of the schedule.

Dynamic Storage Allocation (DSA) Dynamic storage allocation is the problem of packing given axis-aligned rectangles into a horizontal strip of minimum height by sliding the rectangles vertically but not horizontally. The DSA problem is $\mathcal{NP}$-complete. Kierstead was the first one to provide a constant factor approximation algorithm for this problem [Kie88]. Later, Gergov improved on this by proposing first a 5 and then a 3-approximation algorithm [Ger96; Ger99]. In [BKK+04], Buchsbaum et al. give a $(2 + \epsilon)$-approximation for the general case of the same problem, and polynomial time approximation schemes for special cases.

Similarly to the strip packing problem, we relate scheduling to DSA by representing each job as a rectangle. The *fixed and contiguous* machine requirement of a job defines again the length on the x-dimension of the rectangle while the processing time defines the length of the y-dimension. The objective of minimizing the height of the strip is translated into minimizing the makespan of the schedule.

Locality

Extending the work in [BDG+15], Lucarelli et al. studied the impact in backfilling scheduling of topological constraints like *contiguity* and *locality* in hierarchical platforms [LMT+15]. They showed that enforcing these constraints can be done at a small cost, and has minimum negative impact on usual metrics such as makespan, flow-time, or stretch. Scheduling under both contiguity and locality constraints can be seen as a special case of the scheduling problem $P \mid set_j \mid C_{\max}$: it suffices to define the set_j so that it includes only allocations that satisfy our constraints. Bleuse et al. [BDL+18; BLT18] introduced a more general model for interference-aware scheduling in large scale parallel platforms. In this work, a 6-approximation algorithm with respect to makespan minimization has been presented for scheduling under contiguity and locality constraints. More specifically, in [BDL+18], they study the effect of a second type of nodes, the input/output nodes, in conjunction with standard computing nodes. This is motivated by the fact that in many current systems, network congestion is a major issue, due to the vast amount of data that are either needed for, or produced from the execution of an application. The work in [BDL+18] is the more closely related to the model in Chapter 3.

2.2 Energy Minimization and Explorable Uncertainty

In this section, we present previous works focused on energy minimization using the speed scaling technique. Then, we talk about the explorable uncertainty setting and we conclude with works that include this setting in scheduling.

2.2.1 Speed Scaling

Offline

Speed scaling is a standard and well-known mechanism to handle energy consumption in computing systems. Since the seminal paper of Yao et al. [YDS95], in 1995, which introduced the speed scaling mechanism to reduce the consumption of CPU energy, a series of papers, e.g. [AAG15; ABK+19; ABL+17; AMS14; BBC+11; BKL+15; BKL+18; BKP07; BLL15; GNS14], and surveys, e.g. [Alb10; Bam16; GHH16], have been published.

Single Machine In [YDS95], each job has to be executed preemptively between its arrival time and deadline by a single variable-speed processor. An offline algorithm (YDS), that is optimal with respect to minimizing the total energy consumption, is proposed. We briefly present algorithm YDS that we use as black box in Chapter 4.

YDS algorithm uses as a key feature the *intensity* of an interval $[t, t']$, which is defined as $\frac{\sum_j w_j}{t'-t}$, where $t < t'$. It then calculates the interval with the biggest intensity, called the critical interval. The jobs that belong in this interval are then scheduled using the intensity as speed and the earliest deadline first policy. Next, it modifies the instance to reflect the deletion of the critical interval by adapting the jobset and the release times and deadlines of the remaining jobs. The algorithm repeats the two steps until all jobs are scheduled.

Multiple Machines Albers et al. [AAG15] study the same problem of dynamic speed scaling but in multi-processor environments with m parallel variable-speed processors, assuming that job migration is allowed at no cost. They solve optimally the offline version of the problem. Bingham and Greenstreet [BG08] proposed a polynomial-time algorithm for the more general setting where the jobs have arbitrary release dates and deadlines. However, this work needs an algorithm for linear programming as a black box. Angel et al. formulate the same problem as a convex program and propose a combinatorial polynomial-time algorithm which is based on finding maximum flows [ABK+19]. Greiner et al. [GNS14] gave a generic reduction, transforming an optimal schedule for the multiprocessor problem with preemption and migration to a schedule with preemption but without migration. In [AMS14] they allow the preemption of jobs but not their migration and they show that the problem is *strongly-*$\mathcal{NP}$-hard even for unit size jobs. The problem is shown to be $\mathcal{NP}$-hard even for instances where jobs share the same release time and deadline. Approximation algorithms are proposed for both cases. In [ABL+17], Albers et al. study the speed scaling setting on a set of heterogeneous processors, where the energy consumption rate is processor-dependent.

Online

Single Machine In the initial work of Yao et al [YDS95] two online algorithms are described for the problem of minimizing the total energy consumption. Firstly, the Average Rate heuristic (AVR) is shown to have a constant competitive ratio, i.e., $2^{\alpha-1}\alpha^\alpha$, for any power function with $\alpha \geqslant 2$ [YDS95]. A lower bound of α^α is stated but not proved in this paper. Bansal et al. [BBC+11] showed that this competitive ratio is essentially tight. They provide a nearly matching lower bound of $\frac{((2-\delta)\alpha)^\alpha}{2}$, where δ is a function of α that approaches zero as α approaches infinity. Secondly, the Optimal Available (OA) heuristic is introduced but not analyzed in the original work [YDS95]. Essentially, OA runs the optimal YDS algorithm every time a new job arrives, keeping in mind to reduce the workload of the scheduled jobs by the amount of work already executed. Bansal et al. [BKP07] gave a tight α^α bound on the competitive ratio of OA with respect to energy. Furthermore, they propose a new online algorithm (BKP) that is e-competitive with respect to maximum speed, and $2\left(\frac{\alpha}{\alpha-1}\right)^\alpha e^\alpha$-competitive with respect to energy, which is lower than the ratio of OA for any $\alpha \geqslant 5$. They also show

that no deterministic online algorithm can have a better competitive ratio with respect to maximum speed. We briefly present here algorithms AVR and BKP that we use as black box in chapter 4.

AVR algorithm uses the concept of density. The density is the lowest constant speed the scheduler must use for a job in order to meet its deadline, and defined as $\delta_j = \frac{w_j}{d_j - r_j}$. It then sets the speed of the processor to be the sum of the densities of all jobs that are released and not yet finished. Again the earliest deadline first policy is used to choose among available jobs.

BKP algorithm estimates the speed at which YDS would work at any time, based on the knowledge of task that have already arrived. It then schedules the unfinished job with the earliest deadline using e times that speed.

Multiple Machines For the online setting, Albers et al. [AAG15] extend the two algorithms proposed by Yao et al. [YDS95] into OA(m) and AVR(m) for multiple machines. They show that OA(m) is α^α-competitive and that AVR(m) achieves a competitive ratio of $2^{\alpha-1}\alpha^\alpha + 1$. Various constraints for speed scaling in multiple machines have been studied. In [ABL+17], the authors analyze the online AVR algorithm in the setting of heterogeneous machines.

2.2.2 Explorable Uncertainty

Kahan [Kah91] was the first to formalize the notion of *explorable uncertainty*. In his work, there is a set of elements with uncertain values that lie in a closed interval and the operation that allows to obtain the exact value of an element is called a *query*. He applied this framework in the context of selection problems. Since then, a series of problems have been studied (e.g. see the survey [EH15]). For instance, in [FMP+03; GSS11; Kah91], the problem of finding the k-th smallest value in a set of uncertain intervals has been studied. In [OW00], Olston and Widom study caching problems in distributed databases. Other problems that have been studied include, the shortest path problem [FMO+07], the knapsack problem [GGI+15] and the minimum spanning tree problem [HEK+08; MMS15]. The goal in most of these works is the minimization of the number of queries to guarantee an exact optimum solution. In [OW00], the trade-off between the number of queries and the precision of the returned solution has been studied.

Scheduling Contrary to the previous approaches, queries in scheduling are executed directly on the machine running the jobs and so it is important to balance the time spent on queries and the time spent on the actual execution of jobs. In this setting, as long as the query needs computational power to reveal extra knowledge, it is more

appropriate to call it a test. In what follows the words query and test are equivalent. Scheduling under the concept of explorable uncertainty is studied in [ABK+18; AE20; DEM+18]. In [DEM+18], the authors consider the problem of scheduling jobs on a single machine when the cost of each query/test is unitary. In their model, the uncertain information concerns the processing time of each job for which an upper bound is known in advance. It is possible to learn the exact processing time by querying at the price of a unit cost. If a job is executed without a query, then its execution time is equal to its upper bound. In [AE20], the authors extend the problem to non-uniform job-dependent testing times and they present new competitive algorithms, both deterministic and randomized, for different objectives of the problem, i.e. minimizing the makespan and minimizing the sum of completion times. In [ABK+18], a single-machine scheduling problem is considered, where given a set of n unit-time jobs, and a set of k unit-time errors, the objective is to reveal n error-free timeslots with the minimum number of queries. The authors present both lower bounds and asymptotically tight upper bounds for different variants of the problem.

2.3 Total Completion Time Minimization and Uncertain Predictions

In this section, we briefly discuss related work to the total completion time minimization objective for both the offline and online cases. In case release times are available, we distinguish between the two setting by writing r_j and $online-r_j$ respectively. In addition, we present past works that take advantage of the uncertain predictions and conclude with those that introduce this framework to scheduling problems.

Offline

A classic scheduling problem is to minimize the sum of completion times in a single machine, i.e. $1 \mid\mid \sum_j C_j$. This problem can be solved optimally by following the *shortest processing time first (SPT)* rule. We briefly present here algorithm SPT that we use in Chapter 5.

SPT algorithm assigns the job with the smallest processing time among all unassigned jobs, whenever a machine is free.

Often a weight, w_j, is related to each job and the objective is to minimize the total weighted completion time, $\sum_j w_j C_j$. This variation can also be solved optimally, by following Smith's rule, i.e. jobs are scheduled in ascending order of the ratios p_j/w_j [Smi56]. Note that the same problem where jobs are allowed to have some restricted class of precedence constraints (in-tree, out-tree, etc) can also be solved optimally. However, it

becomes *strongly-$\mathcal{NP}$*-hard for arbitrary precedence constraints. In case all weights are equal to 1, Smith's rule coincides with the SPT rule.

When release times are introduced, the problem becomes $\mathcal{NP}$-hard for the non-preemptive case. Phillips, Stein and Wein were the first to explore the weighted completion time objective from the approximation algorithm point of view [PSW98]. Following this work, constant factor approximation algorithms [HSS+97; Sch96], inapproximability results [HSW01] and polynomial time approximation schemes [ABC+99; SW00] for several variants of the problem have appeared. If preemption is allowed, the *shortest remaining processing time (SRPT)* algorithm creates an optimal schedule. We briefly present here algorithm SRPT that we use in Chapter 5.

SRPT algorithm schedules that ready job with the smallest remaining processing time, at any moment of time. SRPT is the preemptive version of SPT, also known as Baker's rule. Note also that SRPT algorithm works for online instances.

For parallel machines, the SPT rule remains optimal for the objective of minimizing the sum of completion times with or without preemptions and all jobs are available at the same time. However, if weights are related to jobs, the problem $P \mid\mid \sum_j w_j C_j$ is *strongly-$\mathcal{NP}$*-hard. The problem with preemption and release times, $P \mid pmtn, r_j \mid \sum_j C_j$, is $\mathcal{NP}$-hard for any number of machines greater or equal to 2 [DL93]. There exist polynomial time approximation schemes for both versions of the problem [ABC+99]. We briefly present here algorithms SPT and SRPT that we use in Chapter 5.

Online

When jobs become available over time, we can distinguish two cases. In the clairvoyant schedule, the processing time, p_j, of a job becomes available when the job arrives, while in the non-clairvoyant schedule on the other hand, only the existence of a job is revealed to algorithm which finds out the processing time of the job only after allocating p_j units of time to it and hence the job has finished.

Clairvoyant Lu et al. [LSS03] consider the problem of scheduling jobs online on a single machine and on identical machines with the objective to minimize total completion time. They give a general 2-competitive algorithm for the single machine problem. They also show that the algorithm is 2α-competitive for the problem on identical machines where α is the performance ratio of the SRPT rule for the preemptive relaxation of the problem. Hoogeveen and Vestjens [HV96] gave different 2-approximation algorithms for $1 \mid online - r_j \mid \sum_j C_j$. Phillips et al. [PSW98] presented another algorithm for $1 \mid r_j \mid \sum_j C_j$, which can be slightly modified to work with online release dates. This algorithm converts a preemptive schedule into a non-preemptive one of objective function value at most twice that of the preemptive schedule. Chekuri et al. [CMN+01]

obtain an optimal randomized online algorithm for the same problem that beats a lower bound for deterministic online algorithms. They also consider extensions to the case of parallel machine scheduling, obtaining a $(3 - \frac{1}{m})$-competitive algorithm for $P \mid online - r_j \mid \sum_j C_j$. Vestjens [Ves97] proved a universal lower bound of 1.309 for the competitive ratio of any deterministic online algorithm for $P \mid online - r_j \mid \sum_j C_j$. In the preemptive case, the currently known lower bound is $\frac{22}{21}$, also given by Vestjens.

Non-Clairvoyant Motwani, Phillips and Torng [MPT94] were the first to study the non-clairvoyant scheduling problem to minimize average completion time. In their work, non-clairvoyant algorithms are compared to optimal clairvoyant ones resulting in both single and multiple machines variants. They show that the *Round Robin (RR)* algorithm has a performance ratio of $(2 - \frac{2}{n+1})$ which is optimal for deterministic, non-clairvoyant algorithms. Here n represents the number of jobs. When m machines are available and all jobs are available at time 0, they show that RR is $(2 - \frac{2m}{n+m})$-competitive for the non-clairvoyant setting. In addition, the authors give both deterministic and randomized lower bounds when jobs have release dates. Garg et al. give a 10-competitive deterministic online algorithm for minimizing the average weighted completion time on parallel machines with both release dates and precedence constraints, in the online non-clairvoyant setting [GGK+19]. Based on this work, in Chapter 5, we analyze RR and give a 4-competitive analysis for minimizing the average completion time on a single machine without precedence constraints. In a very recent work [MV22], Moseley and Vardi study round robin's performance for the ℓ_p-norm of the completion times when scheduling n preemptive jobs on a single machine. For the version of the problem without release times on a single machine, they prove RR's approximation ratio to be $\sqrt[p]{p+1}$, while when jobs arrive over time, RR's competitive ratio is shown to be at most 4 for any $p \geqslant 1$.

2.3.1 Uncertain Predictions

Medina and Vassilitskii [MV17] and Lykouris and Vassilitskii [LV18] were the first to introduce predictions to improve the performance of online algorithms. The first one use a predictor oracle to improve revenue optimization in auctions by setting a good reserve (or minimum) price while the second develops the novel framework even more by introducing the notions of consistency and robustness. In the second work, the online caching problem with predictions is considered. Following this work, a series of learning augmented results appeared in various fields. Problems such as caching [ACE+20; JPS20; Roh20], ski-rental [AGP20; BMS20; GP19; WL20], clustering [DIR+20] and others [HIK+19; LLM+20; LMH+21; Mit20] have been studied under the new setting.

Scheduling More related to scheduling, Bamas et al. [BMR+20] consider the prob-

lem of speed scaling with predictions and in a different work Bamas et al. [BMS20] show how to incorporate predictions that advice the online algorithm about the next action to take using the primal-dual schema. Moreover, Purohit et al. [PSK18] applied this novel setting to the classic ski rental problem and the non-clairvoyant scheduling on a single machine without release times for the objective of minimizing the sum of completion times. For the same problems, Wei and Zhang [WZ20] provide a set of non-trivial lower bounds for competitive analysis using machine-learned predictions. Focused on the single machine non-clairvoyant scheduling problem, Im et al. [IKQ+21] propose a new error measure for prediction quality and design scheduling algorithms under this measure. The work by Purohit et al. [PSK18] deals with the problem $1 \mid non-clair \mid \sum_j C_j$. In Chapter 5, we extend their work by studying $P \mid non-clair \mid \sum_j C_j$ and $1 \mid online-r_j, non-clair \mid \sum_j C_j$ which are left open.

3 Topological Constraints

In this chapter, we discuss the problem of scheduling malleable jobs under topological constraints. Bleuse et al. [BDL+18] introduced a general model for interference-aware scheduling in platforms where machines form a line. They considered two different types of communications: the flows induced by data exchanges during computations and the flows related to Input/Output operations. Rather than taking into account these communications explicitly, they restrict the possible allocations of a job by external topological constraints, that aim at preventing data movement and thus reducing energy consumption. In their work, jobs are considered to be rigid: a job requires a specific number of machines in order to be executed. Here, we first adopt the same framework for the platform and the aforementioned topological constraints on the line topology. We show that there is no polynomial time approximation algorithm under the rigid setting with ratio smaller than $3/2$, unless $\mathcal{P} = \mathcal{NP}$. Then, we focus on the malleable setting. We show that in the proportional-malleable setting, where the work of every job remains constant independently of the number of machines on which it is executed, the scheduling problem is $\mathcal{NP}$-hard even in the case where the maximum number of machines is the same for all the jobs. Then, we propose a 2-approximation algorithm for this case. Furthermore, we present an approximation algorithm solving the more general case where the maximum number of machines is job-dependent.

3.1 Formulation of the problem

We model the platform by distinguishing two kinds of nodes: a set P^C of m^C nodes dedicated to computations, and a set $P^{I/O}$ of $m^{I/O}$ nodes that are entry points to a high performance file system. Let $P = P^C \cup P^{I/O}$ and $m = m^{I/O} + m^C$. Usually, $m^{I/O} << m^C$. We assume that each node has a specific functionality: it can either be a computing or an I/O node. Furthermore, we suppose that any computing or I/O node is dedicated to one application throughout its execution, meaning that two jobs cannot use the same node simultaneously.

The network topology considered in this work is the line. This is an interesting topology for the two following reasons. First of all, it is a basic case of higher dimensional topologies and as such it provides lower bounds for the more complex ones. In addition, it retains the attributes of real-life systems which use direct topologies and can be projected in a single dimension, like mesh or $3D$-torus. In a line topology, all nodes (computing and I/O) form a single connected component, each one connected to two other nodes, except the two nodes in the extremities. We assume that the localisation

of every node within the topology is known. In lines, this can be very easily done, by numbering the nodes from left to right.

We see applications as jobs which are queued in a set $\mathcal{J}$. The total number of jobs is n. We distinguish two models with respect to the computing need of a job.

1. In the *rigid* model a job $j \in \mathcal{J}$ requires a fixed number of computing nodes $q_j \leqslant m^{\mathrm{C}}$. The processing time is also fixed, denoted by p_j.

2. In the *malleable* model a job $j \in \mathcal{J}$ asks for a number of computing nodes Q_j, and the scheduler can decide the number of computing nodes $q_j \leqslant Q_j$ to be used for its execution. Each job j has a required execution load, denoted by w_j. The exact processing time of the job j depends on the number of assigned computing nodes. Let $f : \mathbb{N} \to \mathbb{R}$ be a speed-up function. The processing time of j is defined as $p_j = w_j f(q_j)$. A common assumption in parallel computing is that the jobs are *monotonic* [MRT07], that is their processing time is non-increasing when more computing nodes are used, while their total work (execution load plus communication overhead due to the parallelization) is non-decreasing. This is the case when the speed-up function is non-increasing and convex. In this paper, we consider two cases. In the *generalized-malleable model*, the function f is an arbitrary convex non-increasing function. In the *proportional-malleable model*, the total work does not depend on the number of computing nodes assigned to it: $f(q_j) = \frac{1}{q_j}$ and hence $p_j = \frac{w_j}{q_j}$.

In any of the above cases, let $P^{\mathrm{C}}(j)$ be the set of computing nodes assigned to the job $j \in \mathcal{J}$ by the scheduler.

In addition, jobs have a demand for a specific I/O node, denoted as $P^{I/O}(j)$. As mentioned before, applications need to read/write data to the disk. Usually, I/O nodes are the entry points to a high performance file system. Lustre, implemented in the BlueWaters platform, is an example of such a distributed file system. The total address range of the file system is divided in stripes and each I/O node is responsible for a stripe. As a result, applications which know where their data is stored, can ask for the specific I/O node.

Due to the parallelization of the HPC jobs, all the parts of a job need to communicate with each other to complete the execution. We refer to this kind of data flows in the network as *computational communications*. Furthermore, in general, jobs that need to run on HPC platforms are computationally demanding and one reason is the great volume of data they need to process. Specifically, each job needs to read the data from the disk when it starts its execution and write data to the disk once it finishes. We refer to this kind of data flows as *I/O communications*. Given the direct topology of the line, each machine is occupied when traffic needs to pass "through" itself in order to arrive at the destination. If this machine is allocated to a different job, then we have the undesirable effect of delaying the completion time of one job in order to handle the traffic from a different job. In order to avoid both the aforementioned data flows, we

use the following definitions introduced in [BDL+18; BLT18] to restrict the number of possible allocations of a job.

▶ **Definition 3.1.** An allocation for a job j is said to be *contiguous* if and only if the nodes of the allocation form a contiguous range with respect to the nodes' ordering. ◀

▶ **Definition 3.2.** An allocation for a job j is said to be *local* if and only if the node $P^{I/O}(j)$ is adjacent to the computing nodes in $P^{C}(j)$, with respect to the underlying topology. ◀

Note that, Bleuse et al. [BDL+18] presented a 6-approximation algorithm for the problem of minimizing the makespan with rigid jobs under the contiguity and locality constraints. However, these constraints have not been studied in the context of scheduling malleable jobs, which is the main subject of this chapter.

Given the overhead of distant communications, we may add a new kind of locality by introducing a limit on the number of machines that may be used for the execution of the jobs. This limitation is parameterized by a common resource requirement $Q = Q_j$ for all jobs in $\mathcal{J}$ in the malleable model. The value of Q is chosen based on the size and the structure of the platform. We call instances satisfying this kind of locality as *uniform* instances. Note that if $Q = m^{C}$, then the scheduling problem is trivial in the proportional-malleable model, since the total work is constant and thus all jobs will be assigned the maximum number of computing resources. However, for smaller values of Q, the problem becomes $\mathcal{NP}$-hard (Section 3.2).

Our objective is to minimize the maximum completion time among all jobs (i.e., the makespan of the schedule) while enforcing the *contiguity* and the *locality* constraints.

Our Contribution

In this chapter, we consider the problem of scheduling malleable jobs with respect to *contiguity* and *locality* constraints in a line topology, and we give the first complexity and approximability results for it.

In Section 3.2, we present complexity results for both the rigid and the proportional-malleable models, implying also the complexity of the generalized-malleable model. We show that the problems are $\mathcal{NP}$-hard even in very restricted cases. We also show that, for any $\epsilon > 0$, there is no approximation algorithm with ratio $\frac{3}{2} - \epsilon$ for the problem of scheduling rigid jobs with respect to contiguity and locality constraints, unless $\mathcal{P} = \mathcal{NP}$. This result reduces the approximability gap for the rigid model for which a 6-approximation algorithm is known [BDL+18].

In Section 3.3, we first deal with the proportional-malleable model in uniform instances and we propose a novel polynomial-time 2-approximation algorithm. Then, in Section 3.4 we present an approximation algorithm for the generalized-malleable problem. This algorithm is analyzed in a computational way and it achieves an approximation ratio that depends on the function f.

3.2 Complexity

In this section, we propose reductions to classify special restrictive versions of our problem in complexity classes. In the following theorems, we will use the *Partition* problem which is defined as follows: given a finite set $S = \{a_1, a_2, \ldots, a_k\}$ of k positive integers, the objective is to decide whether there is a subset $S' \subset S$ such that $\sum_{i \in S'} a_i = \sum_{i \in S \setminus S'} a_i$.

The problem of scheduling rigid jobs under contiguity and locality constraints is shown to be *strongly-$\mathcal{NP}$*-hard by Bleuse et al. in [BDL+18]. In the following theorem we provide an inapproximability result for this problem.

▶ **Theorem 3.3.** Unless $\mathcal{P} = \mathcal{NP}$, there is no polynomial time approximation algorithm having a guarantee of $3/2 - \epsilon$ for the problem of scheduling rigid jobs with $p_j = 1$ under contiguity and locality constraints, for any $\epsilon > 0$. ◀

Proof. We will prove the inapproximability result by a reduction from a special case of the *Partition* problem. In the *Partition-Pairs* problem, we are given two sets $A = \{\alpha_1, \alpha_2, \ldots, \alpha_k\}$ and $A' = \{\alpha'_1, \alpha'_2, \ldots, \alpha'_k\}$, each one containing k elements. Let $S = A \cup A'$. Each element $\alpha_i \in A$ (resp. $\alpha'_i \in A'$) has weight $a_i \in \mathbb{Z}^+$ (resp. $a'_i \in \mathbb{Z}^+$). Let $B = \sum_{\alpha_i \in A} a_i + \sum_{\alpha'_i \in A'} a'_i$. The goal is to decide if there is a partition of S into two subsets S' and $S \setminus S'$ such that

- $\sum_{\alpha_i \in S'} a_i + \sum_{\alpha'_i \in S'} a'_i = \sum_{\alpha_i \in (S \setminus S')} a_i + \sum_{\alpha'_i \in (S \setminus S')} a'_i = \frac{B}{2}$, and
- for each $i \in \{1, \ldots, k\}$, the elements $\alpha_i \in A$ and $\alpha'_i \in A'$ are not assigned to the same set, i.e., if $\alpha_i \in S'$ then $\alpha'_i \in S \setminus S'$ and vice-versa.

Note that, if all the weights in A' are set to zero, we still need to find a solution for the *Partition* problem in the set A. Thus, the problem *Partition-Pairs* is $\mathcal{NP}$-complete.

We propose now a transformation from *Partition-Pairs* to our problem as follows:

- $m^C = k \cdot B + \frac{B}{2}$, $m^{I/O} = k$
- the topology is a line starting with B computing nodes followed by one I/O node. This pattern repeats for k times and after the k^{th} I/O node we have the last $\frac{B}{2}$ computing nodes. With respect to this ordering, we refer to the computing nodes as $1, 2, \ldots, m^C$ and to the I/O nodes as $1, 2, \ldots, m^{I/O}$.
- for each $\alpha_i \in A$ (resp. $\alpha'_i \in A'$)), we create a job j_i (resp. j'_i) with $q_{j_i} = B + a_i$ (resp. $q_{j'_i} = B + a'_i$). Both jobs j_i and j'_i require the i^{th} I/O node. All jobs of the created instance have unit processing time. Note that $n = 2k$.

Note that this transformation can be done in polynomial time: it is sufficient to give the number of machines $m = kB + \frac{B}{2} + k$, to assume that the machines are numbered from left to right from 1 to m, and to indicate the k numbers which correspond to the

I/O nodes. We will prove that a solution to *Partition-Pairs* exists if and only if there is a schedule that satisfies all constraints and has a makespan at most 2.

Assume that there is a solution $(S', S \setminus S')$ for *Partition-Pairs*. For each $i \in \{1, 2, \ldots, k\}$, let us denote by y_i (resp. z_i) the job corresponding to the element in $\{\alpha_i, \alpha'_i\}$ which belongs to S' (resp. $S \setminus S'$). Then, we create a schedule for our problem as follows: we schedule the jobs corresponding to elements in S' at time interval $(0, 1]$ and the jobs corresponding to elements in $S \setminus S'$ at time interval $(1, 2]$. Specifically, in the time interval $(0, 1]$, the job y_1 will use the computing nodes $1, 2, \ldots, B + q_{y_1}$, the job y_2 will use $B + q_{y_1} + 1, B + q_{y_1} + 2, \ldots, 2B + q_{y_1} + q_{y_2}$, and so on. In general, the job y_i, $1 \leqslant i \leqslant k$, will use the computing nodes $\sum_{\ell=1}^{i-1}(B + q_{y_\ell}) + 1, \ldots, \sum_{\ell=1}^{i}(B + q_{y_\ell})$. In a similar way, in the time interval $(1, 2]$, the job z_i, $1 \leqslant i \leqslant k$, will use the computing nodes $\sum_{\ell=1}^{i-1}(B + q_{z_\ell}) + 1, \ldots, \sum_{\ell=1}^{i}(B + q_{z_\ell})$.

The created schedule satisfies the contiguity constraint. By the construction of the solution $(S', S \setminus S')$, the jobs scheduled in the time interval $(0, 1]$ require in total $\sum_{\ell=1}^{k}(B + q_{y_\ell}) = Bk + \frac{B}{2}$ computing nodes. Similarly, for the jobs scheduled in the time interval $(1, 2]$. Hence, there are enough computing nodes in each time interval. Moreover, by the construction of the scheduling instance, the i^{th} I/O node is between the computing nodes iB and $iB + 1$, for each $1 \leqslant i \leqslant k$. Since for each job y_i, $1 \leqslant i \leqslant k$, it holds that $\sum_{\ell=1}^{i} q_{y_\ell} \leqslant \frac{B}{2}$, then for the leftmost and the rightmost computing nodes assigned to y_i we have $\sum_{\ell=1}^{i-1}(B + q_{y_\ell}) + 1 = (i-1)B + \sum_{\ell=1}^{i-1} q_{y_\ell} + 1 \leqslant (i-1)B + \frac{B}{2} + 1 < iB$ and $\sum_{\ell=1}^{i}(B + q_{y_\ell}) = iB + \sum_{\ell=1}^{i} q_{y_\ell} \geqslant iB + 1$. Thus, the allocation for y_i is local since it always contains the computing nodes iB and $iB + 1$. The same holds for each job z_i, $1 \leqslant i \leqslant k$. Finally, the length of the created schedule is equal to two.

Conversely, assume now that there is a schedule respecting contiguity and locality constraints of makespan at most 2. Due to the unit processing time of each job, each computing node has to execute exactly two jobs. Hence, the partition is directly derived by assigning jobs that are scheduled in the time interval $(0, 1]$ in the set S' and those scheduled in the time interval $(1, 2]$ in the set $S \setminus S'$. Since the scheduling solution respects the locality constraint, the two jobs j_i and j'_i asking for the i^{th} I/O node are scheduled in a different time interval. Therefore, the elements α_i and α'_i are not in the same subset of the solution of the *Partition-Pairs* problem, and hence this solution is feasible for it.

Note that, the above proof directly implies that there is no $3/2 - \epsilon$-approximation algorithm for our scheduling problem, assuming that $\mathcal{P} \neq \mathcal{NP}$. ■

We now focus on the malleable model and we show that the problem is $\mathcal{NP}$-hard even for the proportional-malleable model and uniform instances.

▶ **Theorem 3.4.** The problem of scheduling malleable jobs with respect to contiguity and locality constraints is $\mathcal{NP}$-hard even in the proportional-malleable model and uniform instances. ◀

Proof. We reduce *Partition* to our scheduling problem. We choose $Q \in \mathbb{Z}^+$ and $B > 0$ such that $2BQ = \sum_{i \in S} a_i$. We create a line which consists of $m^C = 2Q$ computing nodes and $m^{I/O} = 3$ I/O nodes. The topology starts with an I/O node, followed by Q computing nodes, the second I/O node, the remaining Q computing nodes and the last I/O node.

For each $i \in S$, we create a "small" job i with $\alpha_i = a_i$, all of them asking for the middle I/O node. Moreover, we create two "big" jobs with $\alpha_j = BQ^2$, each one asking for a different extreme I/O node. All jobs require exactly Q computing nodes, i.e., $Q_j = Q$ for each $j \in \mathcal{J}$.

We will prove that a solution to *Partition* exists if and only if there is a schedule that satisfies all the constraints and has a makespan at most $B(Q+1)$.

Assume that there is a solution $(S', S \setminus S')$ for *Partition*, i.e., $\sum_{i \in S'} a_i = \sum_{i \in S \setminus S'} a_i = BQ$. We create a schedule as follows:

- on the leftmost Q computing nodes, we schedule: (i) the "big" job targeting the left I/O node in the time interval $(0, BQ]$ using $q_j = Q$ and hence a processing time $p_j = \frac{BQ^2}{Q} = BQ$, and (ii) the "small" jobs corresponding to the elements of the set S' (and targeting the middle I/O node) in the time interval $(BQ, BQ + B]$ using for each of them $q_j = Q$ and hence a processing time $p_j = \frac{a_j}{Q}$.

- on the rightmost Q computing nodes, we schedule: (i) the "small" jobs corresponding to the elements of the set $S \setminus S'$ (and targeting the middle I/O node) in the time interval $(0, B]$ using for each of them $q_j = Q$ and hence a processing time $p_j = \frac{a_j}{Q}$, and (ii) the "big" job targeting the right I/O node in the time interval $(B, BQ + B]$ using $q_j = Q$ and hence a processing time $p_j = \frac{BQ^2}{Q} = BQ$.

By construction, the contiguity and the locality constraints are satisfied, while the makespan of the schedule is exactly $B(Q+1)$. Due to the solution of *Partition*, we have that $\sum_{i \in S'} a_i = BQ$, and hence the total processing time of all jobs corresponding to the elements of S' is $\frac{BQ}{Q} = B$, fitting in the assigned time interval. The same argument holds for the jobs corresponding to the elements of $S \setminus S'$. Thus, the created schedule is feasible.

Conversely, given a feasible optimal schedule of makespan $B(Q+1)$, we first need to show that both "big" jobs are necessarily executed using $q_j = Q$ computing nodes. Suppose that a "big" job is executed using $q_j < Q$ computing nodes. Therefore, the load on each of these nodes is at least

$$\frac{BQ^2}{q_j} \geqslant \frac{BQ^2}{Q-1} = \frac{B(Q^2-1)+B}{Q-1} = \frac{B(Q-1)(Q+1)+B}{Q-1} = B(Q+1) + \frac{B}{Q-1}$$

which is strictly greater than $B(Q+1)$, and hence we have a contradiction to the feasibility of the schedule. Moreover, the total processing time of "small" jobs that are executed on the computing node just on the left of the middle I/O node is at most $B(Q+1) - BQ = B$;

similarly, for the computing node which is just on the right of the middle I/O node. Furthermore, the "small" jobs have a total processing time at least $2B$; this happens if all of them use Q computing nodes. We conclude that the set of "small" jobs has been partitioned into two subsets of the same total processing time, and therefore we can construct a solution for the *Partition* problem. ■

In the proof of the previous theorem, if $Q = 1$ then the constructed instance coincides with a special case of the rigid model where each job is executed on exactly one machine (i.e. tasks are sequential). We get the following corollary.

▶ **Corollary 3.5.** The problem of scheduling rigid jobs with respect to contiguity and locality constraints is $\mathcal{NP}$-hard even if $m^C = 2$, $m^{I/O} = 3$ and $q_j = 1$ for each job $j \in \mathcal{J}$. ◀

Finally, we can extend the proof given in [BDL+18] for the rigid model, and get the following theorem. The proof is very similar to the one provided in [BDL+18] and therefore not presented here.

▶ **Theorem 3.6.** The problem of scheduling malleable jobs with respect to contiguity and locality constraints is *strongly-*$\mathcal{NP}$-hard even in the proportional-malleable model with $m^{I/O} = 3$ and $w_j = Q_j$, for each $j \in \mathcal{J}$. ◀

3.3 Proportional Malleable Model

In this section, we propose approximation algorithms for the problem of scheduling malleable jobs with respect to *contiguity* and *locality* constraints. In a *uniform instance* of the Proportional-Malleable Model, each job can be executed by at most Q computing nodes. We denote by $\mathcal{M}_j$ the set of computing nodes (machines) that can participate in the execution of a job j. Let $\mathcal{M}$ be an arbitrary set of machines. Then the average time $\mathcal{LB}_1$ for which a machine from $\mathcal{M}$ has to run is a lower bound on the length of an optimal schedule. That is:

$$\mathcal{LB}_1 = \max_{M \subseteq P^C} \left\{ \frac{\sum_{j | \mathcal{M}_j \subseteq \mathcal{M}} w_j}{|\mathcal{M}|} \right\} \tag{3.1}$$

For each node $i \in P^{I/O}$ we denote by $\mathcal{J}_i$ the set of jobs with $P^{I/O}(j) = i$. Given that each job cannot be allocated on more than Q machines, we obtain a second lower bound.

$$\mathcal{LB}_2 = \max_{i \in P^{I/O}} \left\{ \sum_{j \in \mathcal{J}_i} \frac{w_j}{Q} \right\} \tag{3.2}$$

In total, the lower bound is given by the largest of these quantities, $\mathcal{LB} = \max\{\mathcal{LB}_1, \mathcal{LB}_2\}$.

In this section, we present an algorithm that computes a schedule whose makespan does not exceed $2\mathcal{LB}$. Firstly, we transform the instance in order to create a simplified jobset $\mathcal{J}'$. For each node $i \in P^{I/O}$ we replace each set of jobs $\mathcal{J}_i$ with a single job j', the execution load of which is equal to $A_{j'} = \sum_{j \in \mathcal{J}_i} w_j$. After determining the allocation for a job j' in $\mathcal{J}'$, we will assign all jobs j in $\mathcal{J}$ corresponding to the same I/O node to the same set of computing nodes.

▶ **Proposition 3.7.** $\mathcal{J}$ and $\mathcal{J}'$ have the same lower bound $\mathcal{LB}$. ◀

We introduce at this point an *auxiliary problem* which will help us find a feasible schedule to our initial problem.

Auxiliary problem

The instance consists of a set of malleable rectangles $\mathcal{U} = \{U_1, U_2, \ldots, U_m\}$ and a strip of width W. We designate the bottom left corner of the strip as the origin of the xy-plane, letting the x-axis be the direction of the width of the strip, and the y-axis be the direction of the height. Each rectangle U_i has a fixed area A_i and a given access point (which corresponds to the input/output nodes of our original problem) τ_i, such that $0 \leqslant \tau_1 \leqslant \tau_2 \leqslant \cdots \leqslant \tau_m \leqslant W$. We represent the location of each rectangle U_i in the strip by the coordinate (s_i, y_i) of its bottom left corner and the coordinate (f_i, y_i) of its bottom right corner. We denote the width of a rectangle as $\lambda_i = f_i - s_i$. In this case, the height of the rectangle U_i is equal to $h_i = \frac{A_i}{\lambda_i}$.

We say that the location of the rectangles is valid if the following *conditions* hold:

1. $s_i, f_i \in \mathbb{N}, i = 1, \ldots, m$
2. $1 \leqslant \lambda_i \leqslant Q, i = 1, \ldots, m$
3. $s_i \leqslant \tau_i \leqslant f_i, i = 1, \ldots, m$
4. $s_i \leqslant s_j, \forall\, i < j$
5. $f_i \leqslant f_j, \forall\, i < j$

In correspondence with the initial scheduling problem, condition 1 means that the computing need of a job is integral, while in condition 2, a job must ask for at least one node with the upper limit being Q. Condition 3 guarantees locality for a job. Finally, conditions 4 and 5 ensures jobs are scheduled following the ordering of the I/O nodes.

The objective of the auxiliary problem is to place m rectangles into the strip without intersections, so as to minimize the height H of the strip.

Consider an arbitrary rectangle U_i. In order to satisfy conditions 2, 3, we define the interval $[r_i, d_i]$. Let $r_i = \max\{0, \tau_i - Q\}$ and $d_i = \min\{W, \tau_i + Q\}$. We can see this

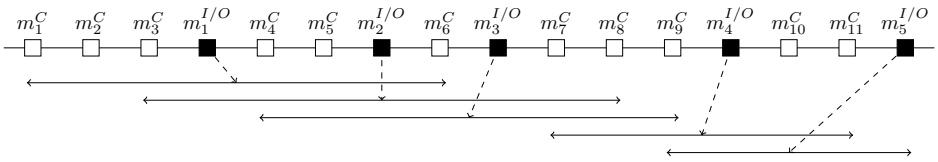


Figure 3.1: Intervals $[r_i, d_i]$ for jobs with for $Q = 3$

interval as the set of computing nodes, where a job i can be scheduled both *locally* and *contiguously*. An example of these intervals is shown in Figure 3.1. Conditions 2 and 3 also imply that in any feasible solution of the auxiliary problem we have $r_j \leqslant s_j < f_j \leqslant d_j$.

In terms of the auxiliary problem, we can re-write lower bounds (3.1) and (3.2) as follows.

$$\mathcal{LB}_1 = \max_{i,j|i<j} \left\{ \frac{\sum_{k|r_i \leqslant r_k \leqslant d_k \leqslant d_j} A_k}{d_j - r_i} \right\} \tag{3.3}$$

$$\mathcal{LB}_2 = \max_i \left\{ \frac{A_i}{Q} \right\} \tag{3.4}$$

Finally, we have that, $\mathcal{LB} = \max\{\mathcal{LB}_1, \mathcal{LB}_2\}$

▶ **Proposition 3.8.** If we do not impose integrality and locality (conditions 1 and 3), the auxiliary problem under conditions 2, 4, 5 has a solution of $H = \mathcal{LB}$. ◀

To create such a solution, we set $\lambda_i = \frac{A_i}{\mathcal{LB}}$ for all $U_i \in \mathcal{U}$. From (3.4) we have $\lambda_i \leqslant Q$. We then place each rectangle on the bottom line, so $y_i = 0$ for all $U_i \in \mathcal{U}$. We determine the x-coordinates of each rectangle according to the following rule. For the first rectangle we set $s_1 = 0$ and $f_1 = \lambda_1$. For the rest of the jobs we set $s_i = \max\{r_i, f_{i-1}\}$ and $f_i = s_i + \lambda_i$.

In the next proof we show that the solution described above and summarized in Algorithm 1, provides a feasible solution to the auxiliary problem with respect to condition 2, 4, 5.

Proof. We will prove by contradiction that $f_i \leqslant d_i$ for all i. Suppose there exists a rectangle U_i such that $f_i > d_i$. Let $j < i$ be the maximal index of a rectangle such that $s_j = r_j$. Then we have $s_k = f_{k-1}$ for all $k = j+1, \ldots, i$. Thus, we have that

$$\sum_{k=j}^{i} \lambda_k > d_i - r_j \tag{3.5}$$

Moreover, for each rectangle U_k, where $k = j + 1, ldots, i$ we have $r_j \leqslant r_k$ and $d_k \leqslant d_i$. From lower bound (3.3) we obtain

$$\mathcal{LB} \geqslant \frac{\sum_{k=j}^{i} A_k}{d_i - r_j} = \frac{\mathcal{LB} \sum_{k=j}^{i} A_k}{\mathcal{LB}(d_i - r_j)} = \frac{\mathcal{LB} \sum_{k=j}^{i} \lambda_k}{d_i - r_j} > \mathcal{LB}$$

where the last inequality follows from (3.5); contradiction. ■

Algorithm 1 places all the rectangles as close as possible to the left edge of the strip. We shift to the right those rectangles for which $f_i < \tau_i$, without changing the order or the location of the other rectangles. Thus, a rectangle can be moved either until it becomes *local*, or until it meets the starting node of the next job. This procedure, which returns a strip S', is described in Algorithm 2.

The solution created by Algorithm 2 may consist of several blocks of jobs. A *block* is a maximal set of rectangles that form a continuous strip of size $\mathcal{LB}$ and consists of:

- a set of *local* rectangles for which $\tau_i \in [s_i, f_i]$,
- a set of rectangles, $\mathcal{L}$ (left), for which $\tau_i > f_i$,
- a set of rectangles, $\mathcal{R}$ (right), for which $\tau_i < s_i$.

Moreover, all left rectangles precede local rectangles, and all right rectangles succeed local rectangles. Notice that sets $\mathcal{L}$ and $\mathcal{R}$ may be empty, while the set of local rectangles must have at least one rectangle. After applying Algorithm 2, it is the local rectangles which prevent other jobs from being local. Therefore, there is always a local rectangle in a block. The structure of a block can be seen in Figure 3.3 (a).

Our main idea is to split the set of rectangles in $\mathcal{U}$ into two subsets, $\mathcal{U}_1$ and $\mathcal{U}_2$, and then pack each of them into a strip of height $\mathcal{LB}$. We start with local rectangles. We number the rectangles in the order as they are carried out in the strip S', resulted by Algorithm 2. Then, we re-allocate each local rectangle i to the interval $[\lfloor s_i \rfloor, \lceil f_i \rceil]$. Subsequently, we set all odd local rectangles in the first subset $\mathcal{U}_1$ and all even rectangles on the second subset $\mathcal{U}_2$. Each rectangle i has a positive width, and therefore we get that $s_{i+2} \geqslant f_{i+1} > s_{i+1} = f_i$. Since each τ_i is integral, and for each local rectangle we

Algorithm 1: Auxiliary solution

```
1 λ_1 = A_1/LB; y_1 = 0; s_1 = 0; f_1 = λ_1;
2 for i > 1 | U_i ∈ U do
3     λ_i = A_i/LB; y_i = 0;
4     s_i = max{r_i, f_{i-1}}; f_i = s_i + λ_i;
```

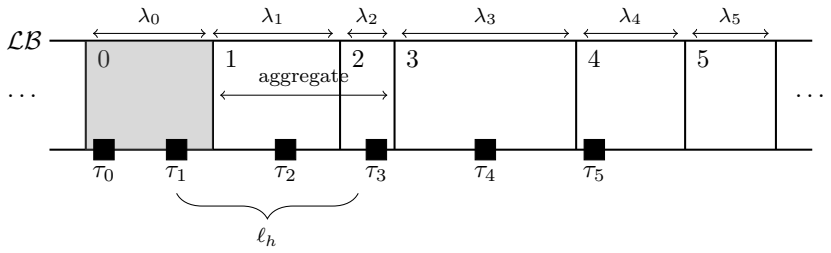


Figure 3.2: Example of creating the first aggregated job of a block in $\mathcal{R}$. The renumbering of nodes and jobs of this step is also depicted. Jobs correspond to the jobs $5-10$ shown in Figure 3.3 (a).

have $\tau_i \in [s_i, f_i]$, we get that $\lfloor s_{i+2} \rfloor \geqslant \lceil f_i \rceil$, and as a result local rectangles from the same subset do not overlap.

For rectangles which are not local, we focus on each block separately. We make rectangles of a block to be *local*, starting by jobs in set $\mathcal{R}$. Rectangles in set $\mathcal{L}$ are handled similarly and therefore the procedure is not explicitly mentioned here.

We call the last local rectangle in the block as R_0. Consider the rectangles in $\mathcal{R} \cup \{R_0\}$. We refer to the access point of the rectangle R_0 as τ_0 and respecting the ordering from left to right we re-number the access points of the rectangles in $\mathcal{R}$. Rectangles' indices follow the same numbering. Figure 3.2 illustrates this notation.

Let $r = |\mathcal{R}|$. We partition the set of rectangles $\mathcal{R}$ into disjoint subsets using the following procedure (see Algorithm 3).

We consider each subset $\mathcal{R}_h$ as an aggregated rectangle R_h. For each aggregated rectangle R_h, we define two quantities. The first one is

$$dist^C = \sum_{U_j \in \mathcal{R}_h} \lambda_j \tag{3.6}$$

Algorithm 2: Create Initial Blocks

```
1 Start with the solution S returned by Algorithm 1;
2 for (i = |J'|; i == 1; i--) do
3     if τ_i > f_i then
4         t = min{(s_{i+1} − f_i), (τ_i − f_i)};
5         f_i += t; s_i += t;
```

which corresponds to the total width of all the rectangles in $\mathcal{R}_h$. Let $\tau_h^{min} = \min_{U_j \in \mathcal{R}_h}\{\tau_j\}$ and $\tau_h^{max} = \max_{U_j \in \mathcal{R}_h}\{\tau_j\}$. The second one is

$$dist^{I/O} = \tau_h^{max} - \tau_h^{min} \tag{3.7}$$

and is related to the distance between the leftmost and the rightmost access points in $\mathcal{R}_h$. We then define the width of the aggregated rectangle R_h as

$$\ell_h = \max\left\{\lceil dist^C \rceil, dist^{I/O}\right\} \tag{3.8}$$

More precisely, we replace each rectangle $U_j \in \mathcal{R}_h$ of width λ_j and height $\mathcal{LB}$ with a rectangle of width ℓ_h and height $\lambda_j \mathcal{LB}/\ell_h$, and put them on top of each other. As a result, we get a rectangle of width ℓ_h and of height $dist^C \mathcal{LB}/\ell_h$, which is no more than Q.

At this point, we are ready to make rectangles in $\mathcal{R}$ both *local* and *integral*. We place each aggregated rectangle R_h in the interval $[\tau_h^{min}, \tau_h^{min} + \ell_h]$. For each rectangle, we also need to choose one of the subsets $\mathcal{U}_1$ or $\mathcal{U}_2$.

This choice is based on the placement of the previous rectangle. Rectangles R_0 and R_1 are always assigned to different sets based on the choice for R_0. If the maximum in ℓ_{i-1} is given by $\lceil dist^C \rceil$ and $R_{i-1} \in \mathcal{U}_1$, assign R_i to $\mathcal{U}_2$ (resp. if $R_{i-1} \in \mathcal{U}_2$, assign R_i to $\mathcal{U}_1$). If the maximum in ℓ_{i-1} is given by $dist^{I/O}$ and $R_{i-1} \in \mathcal{U}_1$ (resp. $\mathcal{U}_2$), assign R_i to $\mathcal{U}_1$ (resp. $\mathcal{U}_2$). Similarly, we impose locality in $\mathcal{L}$.

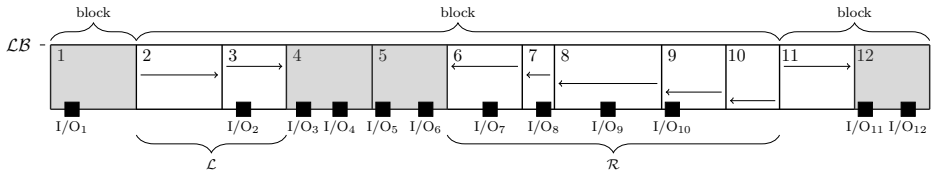
▶ **Lemma 3.9.** The aggregated rectangles do not overlap. ◀

Proof. Since all rectangles in $\mathcal{R}$ are shifted to the left and all rectangles in $\mathcal{L}$ are shifted to the right, the shifted rectangles from different blocks do not overlap. Let U_l be any left aggregated rectangle and U_r to be any right aggregated rectangle. We have $f_l = \tau_l^{max} \leqslant \tau_0 \leqslant \tau_r^{min} = s_r$. It follows that two rectangles U_l, U_r with $U_l \in \mathcal{L}$ and $U_r \in \mathcal{R}$ from the same block do not overlap.

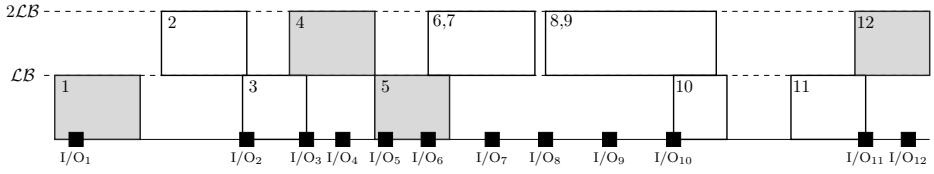
Now we show that rectangles in $\mathcal{R}$ from one block do not overlap. Let $\ell_{h-1} =$

Algorithm 3: Create Aggregated Rectangles

```
1 Put h = 0; k = 1;
2 while k ⩽ r do
3     set i = k, h = h + 1, R_h := ∅;
4     while Σ_{j=i}^{k} λ_j ⩽ Q and τ_k − τ_i ⩽ Q do
5         set R_h = R_h ∪ {U_k}; k = k + 1;
```



(a) Schedule returned by Algorithm 2. Structure of a *block*.



(b) The final schedule after imposing locality and integrality

Figure 3.3: Example of the transformation from a partial solution return by Algorithm 2 to a *contiguous*, *local*, *integral* feasible schedule of our algorithm. Input/Output nodes are depicted by a black square. Computing nodes are not shown. Jobs and Input/Output nodes follow the same numbering. Jobs in gray are *local* while jobs in white need to be reallocated.

$\tau_{h-1}^{max} - \tau_{h-1}^{min}$. Since $\tau_{h-1}^{max} < \tau_h^{min}$ the rectangles J_{h-1} and J_h do not overlap. Now, let $\ell_{h-1} = \sum_{j \in R_{h-1}} \lambda_j$. In this case, rectangles J_h and J_{h-1} belong to different sets and do not overlap. It remains to show that the rectangle J_h does not overlap with previous rectangles. Let U_j be the first rectangle in R_h. From the execution of Algorithm 3, we have $\ell_{h-1} + \lambda_j > Q$. Hence, $f_j - \min_{i \in R_{h-1}} s_i > Q$. Taking into account that $\tau_j = d_j - Q \geqslant f_j - Q$, we obtain

$$\tau_h^{min} = \tau_j \geqslant f_j - Q > \min_{i \in R_{h-1}} s_i = s_{h-1} \geqslant \tau_{h-1}^{min} > \tau_{h-2}^{max}$$

Similarly, rectangles in $\mathcal{L}$ from one block do not overlap. ■

▶ **Lemma 3.10.** The aggregated rectangles and the local rectangles do not overlap. ◀

Proof. We prove this result for jobs in $\mathcal{R}$. Let R_i be the first aggregated rectangle that belongs to the same set as R_0. It follows that $\ell_{i-1} = \sum_{j \in R_{i-1}} \lambda_j$. As shown in Lemma 3.9 we have $\tau_i^{min} \geqslant s_{i-1} \geqslant f_0$. Since τ_i^{min} is integer, we have $\tau_i^{min} \geqslant \lceil f_0 \rceil$ and rectangles R_0 and R_i do not overlap. Similarly, we can prove this result for jobs in $\mathcal{L}$. ■

Following the procedure above, we can find a solution of height $2\mathcal{LB}$ to the auxiliary problem which partitions the rectangles into two subsets, and packs them in two strips of height $\mathcal{LB}$ without intersections.

Now, consider the topology of an instance of the scheduling problem. We number the computing nodes in P^C independently from the I/O nodes, respecting their ordering on the line. Furthermore, we set $W = m^C$ and we associate the unit interval $[i-1, i]$ with the computing node i. For each I/O node $j \in P^{I/O}$ we set $\tau_j = 0$ if the I/O node precedes all computing nodes, $\tau_j = m^C$ if the I/O node follows all computing nodes and $\tau_j = k$ if the I/O node is located between the computing nodes k and $k+1$. For the j-th I/O node we create a rectangle R_j such that its area is A_j.

▶ **Lemma 3.11.** Let S be a feasible solution of the auxiliary problem with height H. Then there exists a feasible solution of the scheduling problem with makespan H. ◀

Proof. Let the rectangle U_j has coordinates (s_j, y_j) and (f_j, y_i) for its bottom left corner and its bottom right corner, respectively. We assign the job j corresponding to the j-th I/O node on computing nodes $\{s_j + 1, \ldots, f_j\}$ in the time interval $(y_j, y_j + h_j]$, where $h_j = \frac{A_j}{\lambda_j}$. Let $P_t^{I/O} = \{j \in P^{I/O} | \tau_j = t\}$. If $s_j < t < f_j$ then job j occupies all I/O nodes from $P_t^{I/O}$. If $s_j = t$, job j occupies the node $P^{I/O}(j)$ and all I/O nodes from $P_t^{I/O}$ to the right of the node $P^{I/O}(j)$. If $f_j = t$, job j occupies the node $P^{I/O}(j)$ and all I/O nodes from $P_t^{I/O}$ to the left of the node $P^{I/O}(j)$. Thus, the job is also local due to condition 3 of the auxiliary problem. Suppose that a job i and a job j overlap on some I/O node. Let $i < j$. Since the rectangles R_i and R_j do not overlap we have $\tau_i = f_i = s_j = \tau_j$ due to conditions 4 and 5. But in this case, the job i does not occupy the I/O nodes to the right of the i-th I/O node and the job j does not occupy the I/O nodes to the left of the j-th I/O node. Hence these jobs do not overlap. ■

This directly yields the following result.

▶ **Theorem 3.12.** There exists a polynomial time 2-approximation algorithm for the problem of scheduling malleable jobs with respect to contiguity and locality constraints on the line in the uniform proportional-malleable model. ◀

3.4 Generalized Malleable Model

Bleuse et al. [BDL+18] introduced an integer linear program for the rigid model with respect to contiguity and locality constraints in order to minimize the total load of each node; note that the maximum load over all nodes is a lower bound to the makespan of the schedule. By solving the relaxed version and rounding this solution, they obtain one allocation whose makespan is at most twice the maximum load in the solution returned by the linear program (LP). Having fixed a valid allocation for each job, the problem coincides with the already known Dynamic Storage Allocation problem for which they use an already known 3-approximation algorithm [Ger99] to create a feasible schedule, getting a 6-approximation algorithm for the rigid model.

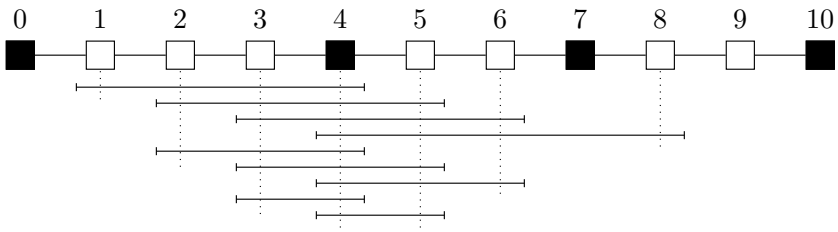


Figure 3.4: Possible valid allocations for a job j asking for 3 computing nodes ($Q_j = 3$) and the 4^{th} node as input/output.

We extend the integer linear program of [BDL+18] as follows. Let $\mathcal{A}_j$ be the set of all potential allocations for each job $j \in \mathcal{J}$. In the malleable model, the set $\mathcal{A}_j$ contains more allocations than the rigid model, as in the former one we have also to decide the number of computing nodes to be used for the execution. Due to the contiguity and the locality constraints, there are $Q_j + 1$ allocations using Q_j computing nodes, Q_j allocations using $Q_j - 1$ computing nodes, and so on. Hence, the number of potential allocations for each job j, $|\mathcal{A}_j| \leqslant \sum_{i=1}^{Q_j}(i+1)$, remains polynomial. Figure 3.4 shows the valid allocations for a job j with $Q_j = 3$.

Each allocation $\ell \in \mathcal{A}_j$ contains a number of computing nodes as well as the required I/O node. Note that, an allocation may include more I/O nodes that will not be used during the execution of j, neither by j nor by any other job due to the locality constraint. By slightly abusing the notation, given an allocation ℓ, we write $i \in \ell$ if the node i is included in ℓ, and we denote by $|\ell|$, the number of computing nodes included in ℓ. Moreover, given an allocation $\ell \in \mathcal{A}_j$ for a job $j \in \mathcal{J}$, we denote by $p_{j\ell} = w_j f(|\ell|)$ the processing time of j if it is executed according to ℓ. Note that the number of different $p_{j\ell}$ is also polynomial. An example of all possible allocations can be seen in Figure 3.4.

For each job $j \in \mathcal{J}$ and allocation $\ell \in \mathcal{A}_j$, we introduce a binary indicator variable $x_{j,\ell}$ which is equal to one if j is executed according to the allocation ℓ, and zero otherwise. Moreover, for each node $i \in P$ (computing and I/O) we introduce a non-negative variable Λ_i which corresponds to the total load of jobs whose assigned allocation includes the node i. Let also Λ be a variable corresponding to the maximum load among all nodes. Then, we consider the following integer linear program which minimizes the maximum load.

$$\begin{aligned}
&\min \Lambda, && && \text{(ILP)}\\
&\text{s.t.} \quad \Lambda \geqslant \Lambda_i && \forall i \in \mathcal{V} && (C_1)\\
&\qquad \Lambda_i \geqslant \sum_{j \in \mathcal{J}} \sum_{\ell \in \mathcal{A}_j} \sum_{i \in \ell} x_{j\ell} p_{j\ell} && \forall i \in \mathcal{V} && (C_2)
\end{aligned}$$

$$\sum_{\ell \in \mathcal{A}_j} x_{j\ell} = 1 \qquad \forall j \in \mathcal{J} \qquad (C_3)$$

$$x_{j\ell} \in \{0,1\} \qquad j \in \mathcal{J}, \ell \in \mathcal{A}_j \qquad (C_4)$$

Constraints (C_1) take the maximum load over all nodes. Constraints (C_2) compute the total load for each node, while Constraints (C_3) ensure that each job is assigned to an allocation. By relaxing the integrity Constraints (C_4) to $x_{j\ell} \in [0,1]$ for each $j \in \mathcal{J}$ and $\ell \in \mathcal{A}_j$, we can solve the corresponding LP in polynomial time. An optimal solution to the relaxed linear program is a lower bound to the makespan of an optimal solution for our problem.

The main differences of the above integer linear program with respect to the one for rigid jobs is that: (i) there is a quadratic number to Q_j of potential allocations for each job j (instead of linear) and (ii) the processing time of j depends on the allocation and the number of computing nodes used by it (instead of a single p_j for all potential allocations).

Consider now an optimal solution of the relaxed linear program. In this solution, let $\tilde{x}_{j\ell}$ be the value of the indicator variable for each job $j \in \mathcal{J}$ and allocation $\ell \in \mathcal{A}_j$, and $\tilde{\Lambda}_i$ be the value of the variable corresponding to the load of node i. In the following, we explain how to round these indicator variables and get an integral allocation for each job $j \in \mathcal{J}$. Let $\bar{x}_{j\ell}$ be the integral value of the indicator variable for each job $j \in \mathcal{J}$ and allocation $\ell \in \mathcal{A}_j$ after the rounding and $\bar{\Lambda}_i$ the corresponding load of node i. We denote by $L_i(j)$ the contribution of job j to the load of node $i \in P$ in solution $\tilde{\Lambda}_i$: $L_i(j) = \sum_{\ell : i \in \ell} \tilde{x}_{j\ell} p_{j\ell}$. Let $\tilde{P}_j = \{i : L_i(j) > 0\}$ be the set of nodes having a positive fractional load for the job j.

Given an allocation $\ell \in \mathcal{A}_j$, we consider the worst case increase of the load of a machine if we decide to schedule j according to ℓ. Specifically, for each node $i \in \tilde{P}_j$ in this allocation ℓ we compute the ratio $\frac{p_{j\ell}}{L_i(j)}$, while the worst case corresponds to the node for which this ratio is maximized. Finally, we decide to schedule j according to the allocation ℓ^* that minimizes this worst case ratio and we set $\bar{x}_{j\ell^*} = 1$. All the other variables for the job j are set to zero, i.e., $\bar{x}_{j\ell} = 0$ for each $\ell \neq \ell^*$. Intuitively, the above procedure aims to choose the allocation for each job $j \in \mathcal{J}$ that increases as little as possible the impact of j on the load of the nodes, without regarding the load of the other jobs.

When a single allocation has been selected for each job, our problem coincides with the Dynamic Storage Allocation problem and we can apply the 3-approximation algorithm proposed in [Ger99]. Algorithm 4 summarizes this procedure.

In what follows, we bound the approximation ratio of Algorithm 4. We initially focus on the rounding procedure (Lines 2–7 of the algorithm). Our analysis is performed for each job $j \in \mathcal{J}$ separately and the approximation ratio of our algorithm depends on the function f and $Q_{\max} = \max\{Q_j, j \in \mathcal{J}\}$. However, the algorithm works for instances with different values of Q_j.

Algorithm 4:

Solve the relaxed version of (ILP)
for *each job* $j \in \mathcal{J}$ **do**
  for *each node* $i \in \{1, \ldots, m\}$ **do**
    $L_i(j) = \sum_{\ell : i \in \ell} \tilde{x}_{j\ell} p_{j\ell}$
  for *each allocation* $\ell \in \mathcal{A}_j$ **do**
    $ratio_j^{\ell} = \max_{i \in \ell, i \in \tilde{P}_j} \left\{ \frac{p_{j\ell}}{L_i(j)} \right\}$
  Choose the allocation $\ell^* = \text{argmin}_{\ell \in \mathcal{A}_j} \{ratio_j^{\ell}\}$
Create a feasible schedule by applying the algorithm proposed in [Ger99] for the Dynamic Storage Allocation problem using the allocations determined by ℓ^*.

The key idea of our analysis is, given a job $j \in \mathcal{J}$, to find a worst-case assignment for the variables $x_{j\ell}$, $\ell \in \mathcal{A}_j$, that maximizes the quantity $\min_{\ell \in \mathcal{A}_j} \{ratio_j^{\ell}\}$: it will maximize the minimal increase of the contribution of task j to the load of a machine between $\tilde{\Lambda}_i$ and $\bar{\Lambda}_i$. Then, any other assignment for the variables $x_{j\ell}$, including the one obtained by solving the relaxed (ILP), will lead to a smaller increase. In order to do this, we create the following *feasibility linear program* for the job $j \in \mathcal{J}$, where α is a constant which corresponds to the value of $\min_{\ell \in \mathcal{A}_j} \{ratio_j^{\ell}\}$ we are searching for.

$$p_{j\ell} \geqslant \alpha \sum_{\ell' : i \in \ell'} x_{j\ell'} p_{j\ell'} \quad \forall \ell \in \mathcal{A}_j, i \in \ell \tag{R_1}$$

$$\sum_{\ell \in \mathcal{A}_j} x_{j\ell} = 1 \tag{R_2}$$

$$x_{j\ell} \geqslant 0 \qquad \forall j \in \mathcal{J}, \ell \in \mathcal{A}_j \tag{R_3}$$

Constraints (R_1) express the ratio between the integral load if allocation ℓ is selected to execute j and the fractional load based on the obtained assignment for a node i, i.e., correspond to the quantity $\frac{p_{j\ell}}{L_i(j)}$. Constraints (R_2) ensure that job j is assigned and guarantees the constraints (C_3) of ILP.

Observe that, the values of $p_{j\ell} = w_j f(|\ell|)$ and $p_{j\ell'} = w_j f(|\ell'|)$ in Constraint (R_1) depend on the same job j. Thus, w_j can be eliminated and the constraint depends only on the number of computing nodes of allocations ℓ and ℓ' (which take value in $\{1, 2, \ldots, Q_j\}$) and the function f. In other words, we obtain the same feasibility linear program for all jobs requiring the same number of computing nodes Q_j, and thus the value of α is not job specific and it depends only on the speed-up function f and the Q_j.

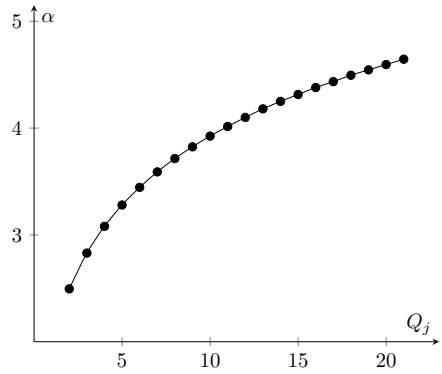


Figure 3.5: Value of α with respect to different Q_j's for the proportional-malleable model.

In order to determine the value of α, we perform a binary search. An infeasible solution to the above linear program implies that it is not possible to have a gap of α and hence we need to choose a smaller value of α. At the end of the binary search procedure, we get the maximum value of α that makes the linear program (R_1)–(R_3) feasible. Then, the following lemma holds.

▶ **Lemma 3.13.** For a job $j \in \mathcal{J}$ and a node $i \in P$, it holds that $p_{j\ell^*} \leqslant \alpha L_i(j)$, where α is the maximum value which makes the linear program (R_1)–(R_3) feasible. ◀

As an example, we calculated the value of α for different values of Q_j in the proportional-malleable model. Figure 3.5 illustrates some of these values.

▶ **Theorem 3.14.** Let $\alpha_{\max}$ be the maximum value over all jobs which makes the linear program (R_1)–(R_3) feasible. Algorithm 4 achieves an approximation ratio of $3\alpha_{\max}$. ◀

Proof. For each job $j \in \mathcal{J}$, let ℓ_j^* be the allocation selected by Algorithm 4 to execute j. Then, for the integral load of the node $i \in P$ we have

$$\begin{aligned}
\bar{\Lambda}_i &= \sum_{j \in \mathcal{J}: i \in \ell_j^*} p_{j\ell_j^*} \leqslant \sum_{j \in \mathcal{J}: i \in \ell_j^*} \alpha_{\max} L_i(j) \\
&= \alpha_{\max} \sum_{j \in \mathcal{J}: i \in \ell_j^*} \sum_{\ell: i \in \ell} \tilde{x}_{j\ell} p_{j\ell} \\
&= \alpha_{\max} \sum_{j \in \mathcal{J}: i \in \ell_j^*} \sum_{\ell \in \mathcal{A}_j} \sum_{i \in \ell} \tilde{x}_{j\ell} p_{j\ell} = \alpha_{\max} \tilde{\Lambda}_i
\end{aligned}$$

By construction, $\tilde{\Lambda} = \max_{i \in P}\{\tilde{\Lambda}_i\}$ is a lower bound the makespan of an optimal schedule for our problem. Then, the theorem follows, since in Line 8 of Algorithm 4 the 3-approximation algorithm for the Dynamic Storage Allocation is used in order to create the final schedule. ■

3.5 Conclusion

In this chapter we studied the makespan minimization problem on the malleable and the rigid models under contiguity and locality constraints. We gave inapproximability results for the rigid model and complexity results for the malleable one. Focusing on the malleable model, we gave approximation algorithms for the proportional uniform setting as well as the generalized one.

4 Explorable Uncertainty

In this chapter, we introduce a model for the speed scaling setting in the framework of explorable uncertainty. In this model, each job has a release time, a deadline and an unknown workload that can be revealed to the algorithm only after executing a query that induces a given additional job-dependent load. Alternatively, the job may be executed without any query, but in that case, its workload is equal to a given upper bound. We study the problem of minimizing the overall energy consumption for executing all the jobs in their time windows. We also consider the related problem of minimizing the maximum speed used by the algorithm. We present lower and upper bounds for both the offline case, where all the jobs are known in advance, and the online case, where the jobs arrive over time. We start with the single machine setting and we finally deal with the more general case where multiple identical parallel machines are available.

4.1 Formulation of the problem

In the classical speed scaling setting, each job j is characterized by a triple (r_j, d_j, w_j), which represents the *release time*, the *deadline* and the *workload* of the job respectively. The workload of j should be entirely executed in the interval $(r_j, d_j]$ which is called its *active interval*. In this chapter, we augment this framework by introducing an *uncertainty* on the workload of the jobs. Here, the work, u_j, is an upper bound rather than an exact value on the actual work needed for the completion of a job. The *exact workload*, $w_j \leqslant u_j$, can be revealed to the algorithm only after executing a *query* (or test) of additional load $t_j \in (0, u_j]$. Hence, in our setting, each job is characterized by a quintuple $(r_j, d_j, t_j, u_j, w_j)$, where w_j is not known before the end of the potential execution of the query. Note that, in the case where the query is not executed, the scheduler is obliged to execute the upper bound of the workload u_j.

We call the above enhanced model as *Query Based Speed Scaling* model (QBSS). The QBSS model is online by nature, since the value of w_j for each job j is revealed only after the potential execution of the query t_j. However, we distinguish between the *offline* and the *online* versions with respect to the classical scheduling setting. In the offline version, the entire input is known in advance, i.e., the total number of jobs to be scheduled, as well as their characteristics, except for the exact loads w_j. In the online version, the input becomes available to the algorithm over time: at time $t = r_j$, a new job j and its characteristics are revealed, except again for its exact load w_j. In other words, the algorithm does not know in advance how many jobs it has to schedule, at which time they will arrive, or what are their characteristics. In both cases, if the exact load of a

job j becomes known at the same time as its other characteristics, then the QBSS model reduces to the classical speed scaling setting, since the scheduler can simply decide whether to make the query for j or not based on the value of $\min\{u_j, t_j + w_j\}$.

This model is inspired by code optimizers which can (potentially) reduce the execution time of program. However, the result cannot be known before the execution of the optimizer. Another application is file transferring over a network, where choosing to compress a file can either reduce its size or not (if the file is already compressed). Other scenarios, such as emergency rooms where an initial check is performed to evaluate the severity of a patient, could be modeled in the same way.

Our contribution

In this chapter, we study an enhanced speed scaling setting (called QBSS), where queries can be optionally executed in the system in order to reveal a more accurate value of the workload of jobs. The main objective is the minimization of the overall energy consumption for executing all the jobs in their time windows (between their release dates and deadlines). We also consider the related problem of minimizing the maximum speed used by the algorithm.

There are two additional questions to answer for each job j in the QBSS model: whether the query will be done or not, and, if yes, how to partition the active interval of the job among the execution of its query and its exact load. Both decisions have a crucial impact on the speed and thus on the consumed energy. For the first question, doing always the query leads to constant approximation algorithms, whereas never doing it leads to unbounded ratios (Section 4.3.1). However, in most cases a better decision can be made by comparing the values of t_j and $\frac{u_j}{\phi}$, where $\phi \approx 1,6180$ is the golden ratio. Note that the optimal offline algorithm has complete knowledge of the instance, including the exact loads. Hence, it can take this decision by comparing u_j and $t_j + w_j$: if $u_j < t_j + w_j$ then the query is not done, otherwise we make the query. For the second question, the algorithm has to determine a *splitting point* $\tau_j = r_j + x(d_j - r_j)$, with $0 < x < 1$ so as $\tau_j \in (r_j, d_j)$, indicating the latest time at which the query has to finish execution and the earliest time at which the exact work of j may start its execution. We introduce the notion of *equal window algorithms* according to which the active interval of a job is split in two equal sub-intervals: the query is executed in the first half, and the exact work in the second half. This is motivated by an instance consisting of a single job, where a different splitting leads to stronger lower bounds (see Lemma 4.4).

In Section 4.3 we study the QBSS model on a single machine. A further discussion, as well as several lower bounds for the offline version of our models are given in Section 4.3.1, where the use of randomization or oracles that answers optimally to one of the questions above are explored. Subsequently in Section 4.3.2, we consider the offline case where all jobs have a common release date and we present a series of results based on different assumptions on the deadlines. Specifically, if all jobs have a common

deadline, we propose the algorithm CRCD which achieves a 2-approximation ratio with respect to maximum speed and a $\min\{2^{\alpha-1}\phi^\alpha, 2^\alpha\}$-approximation ratio with respect to energy. A better analysis is also given for special values of α. Furthermore, in the same section, we consider the case where all deadlines are powers of two and we propose a $(4\phi)^\alpha$-approximation algorithm (CRP2D) with respect to energy. Finally, we extend the previous result to arbitrary deadlines and we obtain an approximation ratio of $(8\phi)^\alpha$ (algorithm CRAD) by rounding down the deadlines of the instance to the closest power of two. In Section 4.3.3, we consider the online case, and we adapt the well-known AVR and BKP online algorithms for the classical speed scaling setting to the QBSS model. The competitive ratio of our algorithms (AVRQ and BKPQ) has an additional multiplicative factor with respect to their version in the classical setting: a factor of 2^α for AVRQ in which the query is made for all jobs, and a factor of $(2+\phi)^\alpha$ for BKPQ in which the execution of the query is decided based on the golden ratio. Note that, BKPQ is also $(2+\phi)e$-competitive with respect to maximum speed.

In Section 4.4 we study the QBSS model on parallel identical machines and we propose a modification of the algorithm AVR(m), which turns out to be $2^\alpha(2^{\alpha-1}\alpha^\alpha+1)$-competitive with respect to energy. Table 4.1 summarizes our results.

Table 4.1: Summary of Our Results

		Energy	
		Lower Bound	Upper Bound
Offline	Oracle	ϕ^α	-
	CRCD	$\max\{\phi^\alpha, 2^{\alpha-1}\}$	$\min\{2^{\alpha-1}\phi^\alpha, 2^\alpha\}$
	CRP2D		$(4\phi)^\alpha$
	CRAD		$(8\phi)^\alpha$
Online	AVRQ	$(2\alpha)^\alpha$	$2^\alpha 2^{\alpha-1}\alpha^\alpha$
	BKPQ	$3^{\alpha-1}$	$(2+\phi)^\alpha 2(\frac{\alpha}{\alpha-1})^\alpha e^\alpha$
	AVRQ(m)	$(2\alpha)^\alpha$	$2^\alpha(2^{\alpha-1}\alpha^\alpha+1)$

4.2 Notations and Preliminaries

We consider a set of n jobs $\mathcal{J}$ which should be executed on a single machine or on a set of m parallel machines $\mathcal{M}$. Each job $j \in \mathcal{J}$ is characterized by a quintuple $(r_j, d_j, t_j, u_j, w_j)$. The scheduler should decide if the initial workload u_j will be executed, or if a query of load $t_j \in (0, u_j]$ will first run in order to reveal the exact (compressed) workload $w_j \leqslant u_j$, which will be executed afterwards. In any case, the whole execution of the job j should be done during its active interval $(r_j, d_j]$. We assume that the *preemption* of the execution of jobs is permitted, while each machine can execute at most one job at

each time. We consider two objectives: the minimization of the maximum speed used, and the minimization of the total energy consumption with respect to the speed scaling mechanism. Then, our goal is to find a feasible preemptive schedule that optimizes one of these objectives.

For a job $j \in \mathcal{J}$, we denote by ω_j the amount of work an algorithm chooses to execute, i.e., $\omega_j = t_j + w_j$ if the query is executed, otherwise $\omega_j = u_j$. Let $\omega_j^* = \min\{u_j, t_j + w_j\}$ be the load executed by the optimal offline algorithm for j. The following lemma describes the relation between the load ω_j^* executed by the optimal solution and the load ω_j executed by an algorithm which decides the execution of the query based on the relation of the quantities t_j and $\frac{u_j}{\phi}$, where ϕ is the golden ratio, i.e., $\phi \approx 1,6180$.

▶ **Lemma 4.1.** Consider an algorithm which decides to make the query for a job $j \in \mathcal{J}$ only if $t_j \leqslant \frac{u_j}{\phi}$. Then, we have $\omega_j \leqslant \phi\omega_j^*$. ◀

Proof. Consider first the case where the algorithm does not perform the query for j, i.e., $t_j > \frac{u_j}{\phi}$ and $\omega_j = u_j$. If $\omega_j^* = u_j$ then $\omega_j = \omega_j^* < \phi\omega_j^*$. If $\omega_j^* = t_j + w_j$ then $\omega_j = u_j \leqslant u_j + \phi w_j = \phi\left(\frac{u_j}{\phi} + w_j\right) \leqslant \phi(t_j + w_j) = \phi\omega_j^*$.

Consider now that the algorithm performs the query for j, i.e., $t_j \leqslant \frac{u_j}{\phi}$ and $\omega_j = t_j + w_j$. If $\omega_j^* = u_j$ then $\omega_j = t_j + w_j \leqslant \frac{u_j}{\phi} + u_j = \left(\frac{1+\phi}{\phi}\right)u_j = \phi u_j = \phi\omega_j^*$. If $\omega_j^* = t_j + w_j$ then $\omega_j = \omega_j^* \leqslant \phi\omega_j^*$. ■

In the classical speed scaling setting without uncertainty, the instance can be described as a set of jobs, each one characterized by the triple (r_j, d_j, w_j). Let $\delta_j = \frac{w_j}{d_j - r_j}$ be the *density* of the job j. The density is an important ingredient in most of the algorithms proposed for this setting as it is related with the speed. Note that the optimal offline solution for the QBSS model coincides with the optimal offline solution in the classical speed scaling setting by using a job (r_j, d_j, ω_j^*) for each job $j \in \mathcal{J}$.

4.3 Single Machine

4.3.1 Lower Bounds

In this section, we will compare the performance of an algorithm in the QBSS offline model, i.e an algorithm which does not know the values w_j, to an optimal algorithm, which knows these values. Our aim is to give lower bounds on the approximation ratio of any algorithm in our setting, for the two objectives that we consider, the minimization of the maximum speed, and the minimization of the total energy. All our results hold for both the single machine case and the multiple machines case, since they will only need to consider a single task. Before introducing our results, let us define a new setting, specifically for instances with one job, which we call the *oracle model*.

Recall that x, $0 < x < 1$, is the fraction of the window $(r_j, d_j]$ in which the query is executed. In other words, in the case where we decide to make the query, then it will be executed in $(r_j, r_j + x(d_j - r_j)]$, while the exact work w_j will be executed in $(r_j + x(d_j - r_j), d_j]$. In the *oracle model*, we suppose the existence of an oracle that can give us the best value of x for the single job of the instance. Therefore, in this model the algorithm needs to take only one decision, i.e. to make or not the query for the job (if the decision is to make the query, the oracle will dictate where to split the window).

Note that the existence of such an oracle is highly improbable, because it translates to knowing the exact load w_j of the job upon its arrival, which conflicts with the setup of our model. The oracle model is however interesting to give lower bounds on the approximation ratio of an algorithm in our setting for two reasons. Firstly, a lower bound on the approximation ratio with the oracle model helps us to better understand the difficulty of our problem. It allows us to see whether the difficulty of a problem is due to the fact that we don't know if it is worthy to do the query or not, or due to the fact that, once we have chosen to do a query, we don't know the exact load w_j before the query has been completed. Of course, a lower bound in the oracle model is also valid in the general model. Secondly, in the following lemmas we mainly create instances of a single task. In the oracle model, once it has been decided that the query will be done, the speed to execute this task will be constant during its whole interval, since this choice minimizes both the maximal speed and the energy due to the convexity of the power function.

▶ **Lemma 4.2.** Any algorithm which never makes the query, can be arbitrarily bad with respect to maximum speed and to energy. ◀

Proof. We consider an instance consisting of a single job j for which $r_j = 0$, $d_j = 1$, $t_j = \varepsilon u_j$, and $w_j = \varepsilon u_j$, with $\varepsilon < 1$ a small positive constant. If the algorithm does not execute the query, then it uses speed $s = \frac{u_j}{d_j - r_j}$, whereas the speed used by an optimal algorithm is $s^* = \frac{\omega_j^*}{d_j - r_j} = \frac{t_j + w_j}{d_j - r_j}$. Concerning the maximum speed, the ratio of such an algorithm is $\frac{s}{s^*} = \frac{u_j}{t_j + w_j} = \frac{1}{2\varepsilon}$, which can be arbitrarily large. Since the speed is constant during the whole interval of size 1, we get that the energy used by the algorithm is $E = s^\alpha$, while the optimal energy is $E^* = (s^*)^\alpha$. The approximation ratio of the algorithm, concerning energy, is thus at least $\frac{E}{E^*} = (\frac{s}{s^*})^\alpha = (\frac{1}{2\varepsilon})^\alpha$, which can be arbitrarily large. ■

▶ **Lemma 4.3.** For any $\epsilon > 0$, there is no deterministic $(\phi - \epsilon)$-approximate algorithm with respect to maximum speed, even in the *oracle model*. Likewise, there is no $(\phi^\alpha - \varepsilon)$-approximate algorithm with respect to the energy, even in the *oracle model*. ◀

Proof. We consider an instance consisting of a single job j active in the interval $(r_j, d_j]$, for which $t_j = 1$ and $u_j = \phi$. Let us consider a deterministic algorithm $\mathcal{A}$ which will use

a single speed s during the whole interval, due to the oracle model. In case where $\mathcal{A}$ does not make the query, then $s \geqslant \frac{u_j}{d_j - r_j} = \frac{\phi}{d_j - r_j}$. We consider in this case that $w_j = 0$. Therefore, the speed of an optimal algorithm is $s^* = \frac{t_j + w_j}{d_j - r_j} = \frac{1}{d_j - r_j}$. The approximation ratio (concerning the maximum speed) of $\mathcal{A}$ is in this case at least ϕ. In case where $\mathcal{A}$ makes the query, then $s \geqslant \frac{t_j + w_j}{d_j - r_j} = \frac{1+\phi}{d_j - r_j}$ and we consider that $w_j = u_j$. We have thus $s^* = \frac{u_j}{d_j - r_j} = \frac{\phi}{d_j - r_j}$. The approximation ratio (concerning the maximum speed) of $\mathcal{A}$ is in this case at least $\frac{1+\phi}{\phi} = \phi$. Therefore, in both cases, the approximation ratio of $\mathcal{A}$, concerning the maximum speed, is at least ϕ.

Note that the speed of both our algorithm and the optimal algorithm is constant during the whole window. Let us consider that $r_j = 0$ and $d_j = 1$. The expected energy of our algorithm will thus be s^α, whereas the energy of the optimal algorithm will be $(s^*)^\alpha$. Concerning the minimization of the energy, the approximation ratio of our algorithm will thus be at least $\frac{E}{E^*} = \frac{s^\alpha}{(s^*)^\alpha} = \left(\frac{s}{s^*}\right)^\alpha = \phi^\alpha$. ■

▶ **Lemma 4.4.** For any $\epsilon > 0$, there is no deterministic $(2 - \epsilon)$-approximation algorithm with respect to maximum speed. Moreover, there is no deterministic $(2^{\alpha-1} - \epsilon)$-approximation algorithm with respect to energy. ◀

Proof. We consider an instance consisting of a single job active in the interval $(r_j, d_j]$, for which $t_j = 1$ and $u_j = 2$. Let $\mathcal{A}$ be a deterministic algorithm. In the case where $\mathcal{A}$ does not make the query, then its speed will be constant during the whole interval and we have that $s = \frac{u_j}{d_j - r_j} = \frac{2}{d_j - r_j}$. In this case the adversary will set $w_j = 0$. Therefore, for the speed of an optimal algorithm we have $s^* = \frac{t_j}{d_j - r_j} = \frac{1}{d_j - r_j}$. The approximation ratio of $\mathcal{A}$ with respect to maximum speed is at least 2, while with respect to energy is at least 2^α.

Let us now consider the case where $\mathcal{A}$ makes the query. Recall that the query is executed in $(r_j, r_j + x(d_j - r_j)]$ and the exact work in $(r_j + x(d_j - r_j), d_j]$. Thus, the speed of $\mathcal{A}$ during the whole first interval is $s_1 = \frac{t_j}{x(d_j - r_j)}$, while during the whole second interval is $s_2 = \frac{w_j}{(1-x)(d_j - r_j)}$. We have two sub-cases with respect to x. If $x \in \left(0, \frac{1}{2}\right]$, then the adversary will set $w_j = 0$, and hence the speed of an optimal algorithm $s^* = \frac{t_j}{d_j - r_j}$ will be constant for the whole interval, while $s_1 \geqslant \frac{2t_j}{(d_j - r_j)}$. In this case, the approximation ratio of $\mathcal{A}$ with respect to maximum speed is at least $\frac{s_1}{s^*} \geqslant 2$, while with respect to energy is at least $\frac{E}{E^*} = \frac{x(d_j - r_j)s_1^\alpha}{(d_j - r_j)(s^*)^\alpha} = \frac{x\left(\frac{t_j}{x(d_j - r_j)}\right)^\alpha}{\left(\frac{t_j}{d_j - r_j}\right)^\alpha} = x^{1-\alpha} \geqslant 2^{\alpha-1}$. If $x \in \left[\frac{1}{2}, 1\right)$, then the adversary will set $w_j = u_j$ having $s^* = \frac{u_j}{d_j - r_j}$. Then the maximum speed used by $\mathcal{A}$ is $s \geqslant \max\{s_1, s_2\} \geqslant s_2 \geqslant \frac{u_j}{(1-x)(d_j - r_j)} \geqslant \frac{u_j}{\frac{d_j - r_j}{2}} = \frac{2u_j}{d_j - r_j}$. In this case, the approximation

ratio of $\mathcal{A}$ with respect to maximum speed is at least $\frac{s_2}{s^*} = \frac{1}{1-x} \geqslant 2$, while with respect to energy is at least $\frac{E}{E^*} = \frac{(1-x)(d_j-r_j)s_2^\alpha}{(d_j-r_j)(s^*)^\alpha} = \frac{(1-x)\left(\frac{u_j}{(1-x)(d_j-r_j)}\right)^\alpha}{\left(\frac{u_j}{d_j-r_j}\right)^\alpha} = (1-x)^{1-\alpha} \geqslant 2^{\alpha-1}$. ■

The next lemma deals with randomized algorithms. We consider that for a given instance I, a randomized algorithm makes the query with a probability ρ_I, and thus does not make it with probability $1-\rho_I$. The approximation ratio of a randomized algorithm is the maximum value, over all instances, of the expected value of the objective function (energy or maximum speed) of the algorithm over the value of an optimal solution. We focus in this chapter on deterministic algorithms, but it is worth noticing that the problem is also difficult, even with a randomized algorithm, and even in the oracle model. As previously, we will use a proof with a single task: the algorithm will only have to choose with which probability it will do the query (if the query is done then the window is divided into two parts optimally, so that the speed is constant during the whole interval).

▶ **Lemma 4.5.** For any $\epsilon > 0$, there is no $(4/3-\epsilon)$-approximate randomized algorithm with respect to maximum speed, even in the *oracle model*. Likewise, there is no $\left(\frac{1}{2}(1+\phi^\alpha)-\varepsilon\right)$-approximate randomized algorithm with respect to energy, even in the *oracle model*. ◀

Proof. We consider an instance consisting of a single job for which $t_j = 1$ and $u_j = w$. Let us consider that the randomized algorithm does the query with a probability ρ. We consider two cases. If $w_j = 0$, then $s^* = \frac{1}{d_j-r_j}$ and $E^* = (d_j-r_j)^{1-\alpha}$. The expected maximum speed of the randomized algorithm is at least $\mathbb{E}[s] = \frac{\rho+(1-\rho)w}{d_j-r_j}$ and the expected energy consumption is at least $(\rho+(1-\rho)w^\alpha)(d_j-r_j)^{1-\alpha}$. Thus, we get an approximation ratio (concerning the maximum speed) of at least $r_1^S(\rho,w) = \rho+(1-\rho)w$ and approximation ratio (concerning the energy consumption) of at least $r_1^E(\rho,w) = \rho+(1-\rho)w^\alpha$. If $w_j = u_j = w$, then $s^* = \frac{w}{d_j-r_j}$ and $E^* = w^\alpha(d_j-r_j)^{1-\alpha}$. The expected maximum speed of the randomized algorithm is at least $\mathbb{E}[s] = \frac{\rho(1+w)+(1-\rho)w}{d_j-r_j}$ and the expected energy consumption is at least $(\rho(1+w)^\alpha+(1-\rho)w^\alpha)\left(d_j-r_j\right)^{1-\alpha}$. In this case we have an approximation ratio (concerning the maximum speed) of at least $r_2^S(\rho,w) = \left(\frac{\rho(1+w)}{w}+(1-\rho)\right)$ and approximation ratio (concerning the energy consumption) of at least $r_2^E(\rho,w) = \left(\frac{\rho(1+w)^\alpha}{w^\alpha}+(1-\rho)\right)$. For arbitrary choice of ρ an approximation ratio is at least $\lambda_S = \max_{w\geqslant 1}\min_{0\leqslant\rho\leqslant 1}\max\{r_1^S(\rho,w), r_2^S(\rho,w)\}$ and $\lambda_E = \max_{w\geqslant 1}\min_{0\leqslant\rho\leqslant 1}\max\{r_1^E(\rho,w), r_2^E(\rho,w)\}$ for the maximum speed and the energy consumption, respectively. It is easy to see that $\lambda_S = \frac{4}{3}$ when $w = 2$ and $\rho = \frac{2}{3}$. To

estimate λ_E we set $w = \phi$. We have:

$$\lambda_E \geqslant \min_{\rho} \max\{\rho + (1-\rho)\phi^\alpha, \frac{\rho(1+\phi)^\alpha}{\phi^\alpha} + (1-\rho)\}$$
$$= \min_{\rho} \max\{\rho + (1-\rho)\phi^\alpha, \rho\phi^\alpha + (1-\rho)\}$$

The last expression reaches a minimum at $\rho = \frac{1}{2}$ and we get $\lambda_E \geqslant \frac{1+\phi^\alpha}{2}$. ■

We note that the lower bound for λ_E can be slightly improved. Indeed, for fixed w we have that λ_E takes its minimum value for $\rho = \frac{w^{2\alpha}-w^\alpha}{(1+w)^\alpha+w^{2\alpha}-2w^\alpha}$ and $\lambda_E(w) = \frac{w^\alpha((1+w)^\alpha-1)}{(1+w)^\alpha+w^{2\alpha}-2w^\alpha}$. Thus, the optimal choice of w depends on α and lies in the range from 1.9 to 2 for $\alpha \leqslant 3$.

▶ **Lemma 4.6.** The competitive ratio of an equal window algorithm is at least 3 with respect to the maximum speed, and at least $3^{\alpha-1}$ with respect to energy. ◀

Proof. We consider an instance consisting of three jobs. We use the quintuple $(r_j, d_j, t_j, u_j, w_j)$ to describe a job. $\mathcal{J} = \{j_1 = (0, 2, 0, 10, 1), j_2 = (1, 3, 1, 10, 0), j_3 = (1, 2, \frac{1}{2}, 10, \frac{1}{2})\}$.

The optimal solution will execute the query and the exact load for all jobs as $t_j + w_j < u_j$ for all j. We obtain the offline optimal schedule for this instance using YDS algorithm. The schedule uses speed 1 for the whole time interval $(0, 3]$. The energy consumed by this optimal schedule is $E^* = (3-0) \cdot 1^\alpha = 3$.

An equal window algorithm will create the following tasks: $\mathcal{J}' = \{j_1^t = (0, 1, 0), j_1^w = (1, 2, 1), j_2^t = (1, 2, 1), j_2^w = (2, 3, 0), j_3^t = (1, \frac{1}{2}, \frac{1}{2}), j_3^w = (\frac{1}{2}, 2, \frac{1}{2})\}$. The final schedule for $\mathcal{J}$ uses speed 0 in the interval $(0, 1]$, speed 3 in the interval $(1, 2]$ and speed 0 in the interval $(2, 3]$. The energy consumed by this schedule is $E = (2-1) \cdot 3^\alpha = 3^\alpha$ and the lemma follows. ■

Note that this last result holds even if we restrict to instances where the optimal algorithm always does the query (since in the example of the proof above both the equal window algorithm and the optimal algorithm always do the query). This shows that, even if an oracle would tell us whether the query should be done or not, the difficulty of splitting the window for each job (query, real workload) is significant.

4.3.2 Offline Model

Common Release, Common Deadline

In this section, we consider that all jobs are released at time 0, and that they have to finish execution at time D. We present CRCD (Algorithm 5), an approximation algorithm with respect to both maximum speed and energy. For each job of our instance, the

Algorithm 5: Common Release, Common Deadline (CRCD)

for *each job* $j \in \mathcal{J}$ **do**
  if $j \in B$, *i.e.,* $t_j \leqslant \frac{u_j}{\phi}$ **then**
    Add $(0, \frac{D}{2}, t_j)$ in set $\mathcal{Q}$;
  if $j \in A$, *i.e.,* $t_j > \frac{u_j}{\phi}$ **then**
    Add $(0, \frac{D}{2}, \frac{u_j}{2})$ in set $\mathcal{W}_1$;
Schedule the jobs in $\mathcal{Q} \cup \mathcal{W}_1$ in an arbitrary order during the interval $\left(0, \frac{D}{2}\right]$ using speed $s(t) = \sum_{j \in \mathcal{Q} \cup \mathcal{W}_1} \delta_j$;
// At time $\frac{D}{2}$ all queries are done;
for *each job* $j \in \mathcal{J}$ **do**
  if $j \in B$ **then**
    Add $(\frac{D}{2}, D, w_j)$ in set $\mathcal{W}^*$;
  if $j \in A$ **then**
    Add $(\frac{D}{2}, D, \frac{u_j}{2})$ in set $\mathcal{W}_2$;
Schedule the jobs in $\mathcal{W}^* \cup \mathcal{W}_2$ in an arbitrary order during the interval $\left(\frac{D}{2}, D\right]$ using speed $s(t) = \sum_{j \in \mathcal{W}^* \cup \mathcal{W}_2} \delta_j$;

algorithm creates two jobs of the classical speed scaling setting. In order to do this, it first partitions the jobs into two subsets A and B, where A and B are defined as follows: $A = \{j \in \mathcal{J} : t_j > \frac{u_j}{\phi}\}$ and $B = \{j \in \mathcal{J} : t_j \leqslant \frac{u_j}{\phi}\}$. By construction, we have that $A \cup B = \mathcal{J}$ and $A \cap B = \emptyset$.

For the jobs in A the algorithm chooses to execute their initial workload without doing a query. Specifically, for each job $j \in A$, it creates two jobs j_1 and j_2 with half the initial workload to be scheduled in the first half and the second half of the initial interval respectively: $(r_{j_1}, d_{j_1}, w_{j_1}) = (0, \frac{D}{2}, \frac{u_j}{2})$ and $(r_{j_2}, d_{j_2}, w_{j_2}) = (\frac{D}{2}, D, \frac{u_j}{2})$. On the other hand, for the jobs in B the algorithm chooses to make the query and hence the exact load of these jobs is revealed once the execution of their query is finished. Specifically, for each job $j \in B$, it creates at time 0 the job $(0, \frac{D}{2}, t_j)$ to be scheduled in the first half of the initial interval. At the end of this first half-interval, the exact workload w_j of j is known, and hence the algorithm creates the job $(\frac{D}{2}, D, w_j)$ to be scheduled in the second half-interval.

▶ **Theorem 4.7.** CRCD (Algorithm 5) achieves an approximation ratio of 2 with respect to maximum speed and of $\min\{2^{\alpha-1}\phi^\alpha, 2^\alpha\}$ with respect to energy. ◀

Proof. The optimal solution for this problem is computed by using the offline optimal

YDS algorithm [YDS95]. Since all jobs are active during the same interval $(0, D]$, the speed during the whole interval is constant and equal to the sum of densities of all jobs. In an optimal solution, the load for each job $j \in \mathcal{J}$ is $\omega_j^* = \min\{u_j, t_j + w_j\}$, and hence its density is $\delta_j^* = \frac{\min\{u_j, t_j+w_j\}}{D} = \frac{\omega_j^*}{D}$. Then, the speed at each time t is $s^* = s^*(t) = \sum_{j\in\mathcal{J}} \frac{\omega_j^*}{D}$ and the total energy consumed by the optimal solution is

$$E^* = \int_0^D (s^*(t))^\alpha dt = D\left(\sum_j \frac{\omega_j^*}{D}\right)^\alpha$$

Algorithm 5 produces a schedule which uses two distinct speeds s_1 and s_2 in the time intervals $(0, \frac{D}{2}]$ and $(\frac{D}{2}, D]$ respectively. For these speeds we have:

$$\begin{aligned} s_1 &= \sum_{j\in Q\cup W_1} \delta_j = \sum_{j\in W_1} \frac{\frac{u_j}{2}}{\frac{D}{2}-0} + \sum_{j\in Q} \frac{t_j}{\frac{D}{2}-0} = \sum_{j\in A} \frac{u_j}{D} + \sum_{j\in B} \frac{2t_j}{D} \\ &\leqslant \sum_{j\in A} \frac{\phi\omega_j^*}{D} + \sum_{j\in B} \frac{2\omega_j^*}{D} \leqslant 2\sum_{j\in\mathcal{J}} \frac{\omega_j^*}{D} = 2s^* \end{aligned}$$

and

$$\begin{aligned} s_2 &= \sum_{j\in W^*\cup W_2} \delta_j = \sum_{j\in W_2} \frac{\frac{u_j}{2}}{D-\frac{D}{2}} + \sum_{j\in W^*} \frac{w_j}{D-\frac{D}{2}} = \sum_{j\in A} \frac{u_j}{D} + \sum_{j\in B} \frac{2w_j}{D} \\ &\leqslant \sum_{j\in A} \frac{\phi\omega_j^*}{D} + \sum_{j\in B} \frac{2\omega_j^*}{D} \leqslant 2\sum_{j\in\mathcal{J}} \frac{\omega_j^*}{D} = 2s^* \end{aligned}$$

where the first inequality in both cases holds by Lemma 4.1, $t_j \leqslant \min\{t_j + w_j, u_j\} = \omega_j^*$ and $w_j \leqslant \min\{t_j + w_j, u_j\} = \omega_j^*$. Hence, Algorithm 5 is 2-approximate with respect to maximum speed.

For the energy consumption of our algorithm we have:

$$E_I = \int_0^{\frac{D}{2}} s_1^\alpha dt + \int_{\frac{D}{2}}^D s_2^\alpha dt = \frac{D}{2}\left(\sum_{j\in A} \frac{u_j}{D} + \sum_{j\in B} \frac{2t_j}{D}\right)^\alpha + \frac{D}{2}\left(\sum_{j\in A} \frac{u_j}{D} + \sum_{j\in B} \frac{2w_j}{D}\right)^\alpha$$

We can now bound the total energy consumption of Algorithm 5. We use two different approaches. In the first approach, we apply the property $x^\alpha + y^\alpha \leqslant (x+y)^\alpha$. Specifically, we have

$$\begin{aligned} E &\leqslant \frac{D}{2}\left(\sum_{j\in A} \frac{u_j}{D} + \sum_{j\in B} \frac{2t_j}{D} + \sum_{j\in A} \frac{u_j}{D} + \sum_{j\in B} \frac{2w_j}{D}\right)^\alpha \\ &= \frac{D}{2}\left(\sum_{j\in A} \frac{2u_j}{D} + \sum_{j\in B} \frac{2t_j+2w_j}{D}\right)^\alpha = 2^{\alpha-1} D\left(\sum_{j\in A} \frac{\omega_j}{D} + \sum_{j\in B} \frac{\omega_j}{D}\right)^\alpha \end{aligned}$$

$$\leqslant 2^{\alpha-1}D\left(\sum_{j\in A}\frac{\phi\omega_j^*}{D}+\sum_{j\in B}\frac{\phi\omega_j^*}{D}\right)^{\alpha}=2^{\alpha-1}\phi^{\alpha}D\left(\sum_{j}\frac{\omega_j^*}{D}\right)^{\alpha}=2^{\alpha-1}\phi^{\alpha}E^*$$

where the second inequality holds by Lemma 4.1.

In the second approach, we bound the energy of the entire interval by twice the maximum energy consumed in one of the two half-intervals. Hence, we have:

$$\begin{aligned}
E &\leqslant 2\cdot\max\left\{\frac{D}{2}\left(\sum_{j\in A}\frac{u_j}{D}+\sum_{j\in B}\frac{2t_j}{D}\right)^{\alpha},\frac{D}{2}\left(\sum_{j\in A}\frac{u_j}{D}+\sum_{j\in B}\frac{2w_j}{D}\right)^{\alpha}\right\}\\
&=D\max\left\{\left(\sum_{j\in A}\frac{\omega_j}{D}+\sum_{j\in B}\frac{2t_j}{D}\right)^{\alpha},\left(\sum_{j\in A}\frac{\omega_j}{D}+\sum_{j\in B}\frac{2w_j}{D}\right)^{\alpha}\right\}\\
&\leqslant D\max\left\{\left(\sum_{j\in A}\frac{\phi\omega_j^*}{D}+\sum_{j\in B}\frac{2\omega_j^*}{D}\right)^{\alpha},\left(\sum_{j\in A}\frac{\phi\omega_j^*}{D}+\sum_{j\in B}\frac{2\omega_j^*}{D}\right)^{\alpha}\right\}\\
&=D\left(\sum_{j\in A}\frac{\phi\omega_j^*}{D}+\sum_{j\in B}\frac{2\omega_j^*}{D}\right)^{\alpha}\leqslant 2^{\alpha}D\left(\sum_{j}\frac{\omega_j^*}{D}\right)^{\alpha}=2^{\alpha}E^*
\end{aligned}$$

where the second inequality holds using Lemma 4.1 and the facts that $t_j \leqslant \min\{u_j, t_j + w_j\} = \omega_j^*$ and $w_j \leqslant \min\{u_j, t_j + w_j\} = \omega_j^*$. The third inequality holds since $\phi < 2$. ■

In what follows in this section, we give a more tight analysis of Algorithm 5 for special values of α based on the following lemma.

▶ **Lemma 4.8.** Let $\alpha \geqslant 2$ and $x \geqslant y$. Then $(x+y)^{\alpha} \geqslant x^{\alpha} + y^{\alpha} + \alpha x^{\alpha-1}y$. ◀

Proof. Set $r = \frac{y}{x} \leqslant 1$. From the binomial series, we have:

$$\begin{aligned}
(x+y)^{\alpha} &= x^{\alpha}(1+r)^{\alpha} \geqslant x^{\alpha}\left(1+\alpha r+\frac{\alpha(\alpha-1)}{2}r^2\right)\\
&= x^{\alpha}+\alpha x^{\alpha-1}y+\frac{\alpha(\alpha-1)}{2}x^{\alpha-2}y^2\\
&\geqslant x^{\alpha}+y^{\alpha}+\alpha x^{\alpha-1}y
\end{aligned}$$

The last inequality holds since $\frac{\alpha(\alpha-1)}{2} \geqslant 1$ and $r \leqslant 1$. ■

▶ **Theorem 4.9.** If $\alpha \geqslant 2$, then Algorithm 5 achieves a competitive ratio of $\max_{r\geqslant 1}\{\min\{f_1(r), f_2(r)\}\}$ with respect to energy, where $f_1(r) = 2^{\alpha-1}\left(1+\frac{1}{r^{\alpha}}\right)$, $f_2(r) = 2^{\alpha-1}\phi^{\alpha}\left[1-\frac{\alpha r^{\alpha-1}}{(r+1)^{\alpha}}\right]$ and $r = \frac{x}{y}$, where $x = \sum_{j\in A}\frac{u_j}{D}+\sum_{j\in B}\frac{2t_j}{D}$ and $y = \sum_{j\in A}\frac{u_j}{D}+\sum_{j\in B}\frac{2w_j}{D}$. ◀

Proof. As before, for the energy of the algorithm, we have:

$$E = \frac{D}{2}\left(\sum_{j\in A}\frac{u_j}{D} + \sum_{j\in B}\frac{2t_j}{D}\right)^{\alpha} + \frac{D}{2}\left(\sum_{j\in A}\frac{u_j}{D} + \sum_{j\in B}\frac{2w_j}{D}\right)^{\alpha}$$

Define $x = \sum_{j\in A}\frac{u_j}{D} + \sum_{j\in B}\frac{2t_j}{D}$ and $y = \sum_{j\in A}\frac{u_j}{D} + \sum_{j\in B}\frac{2w_j}{D}$. For $r \geqslant 1$, let $x = ry$, if $x \geqslant y$, otherwise let $y = rx$. In order to bound this energy consumption, we use two different analyses. For the first analysis, we have:

$$E = \frac{D}{2}x^{\alpha} + \frac{D}{2}y^{\alpha} = \frac{D}{2}x^{\alpha}\left(1+\frac{1}{r^{\alpha}}\right) = \frac{D}{2}\left(\sum_{j\in A}\frac{u_j}{D} + \sum_{j\in B}\frac{2t_j}{D}\right)^{\alpha}\left(1+\frac{1}{r^{\alpha}}\right)$$
$$\leqslant \frac{D}{2}\left(\sum_{j\in A}\frac{\phi\omega_j^*}{D} + \sum_{j\in B}\frac{2\omega_j^*}{D}\right)^{\alpha}\left(1+\frac{1}{r^{\alpha}}\right) \leqslant 2^{\alpha-1}\left(1+\frac{1}{r^{\alpha}}\right)E^* = f_1(r)\cdot E^*$$

where the inequalities follow by using the same arguments as in the second part of Theorem 4.7.

For the second analysis, by using Lemma 4.8, we have

$$E = \frac{D}{2}x^{\alpha} + \frac{D}{2}y^{\alpha} \leqslant \frac{D}{2}\left[(x+y)^{\alpha} - \alpha x^{\alpha-1}y\right]$$
$$= \frac{D}{2}\left[(x+y)^{\alpha} - \alpha\frac{r^{\alpha-1}}{(r+1)^{\alpha}}(x+y)^{\alpha}\right] = \frac{D}{2}\left(1-\alpha\frac{r^{\alpha-1}}{(r+1)^{\alpha}}\right)(x+y)^{\alpha}$$
$$= \frac{D}{2}\left(1-\alpha\frac{r^{\alpha-1}}{(r+1)^{\alpha}}\right)\left(\sum_{j\in A}\frac{u_j}{D} + \sum_{j\in B}\frac{2t_j}{D} + \sum_{j\in A}\frac{u_j}{D} + \sum_{j\in B}\frac{2w_j}{D}\right)^{\alpha}$$
$$\leqslant \left(1-\alpha\frac{r^{\alpha-1}}{(r+1)^{\alpha}}\right)2^{\alpha-1}\phi^{\alpha}E^* = f_2(r)\cdot E^*$$

where the second line holds because

$$x^{\alpha-1}y = r^{\alpha-1}y^{\alpha} = \frac{r^{\alpha-1}}{(r+1)^{\alpha}}[(r+1)y]^{\alpha} = \frac{r^{\alpha-1}}{(r+1)^{\alpha}}(x+y)^{\alpha}$$

while the last inequality follows by using the same arguments as in the first part of Theorem 4.7.

In total, the energy consumption of the algorithm is given by the following relation.

$$E \leqslant \max_{r\geqslant 1}\left\{\min\left\{2^{\alpha-1}\left(1+\frac{1}{r^{\alpha}}\right), 2^{\alpha-1}\phi^{\alpha}\left[1-\frac{\alpha r^{\alpha-1}}{(r+1)^{\alpha}}\right]\right\}\right\}$$ ■

In general, comparing the three ratios $\rho_1 = 2^{\alpha-1}\phi^{\alpha}$, $\rho_2 = 2^{\alpha}$ and $\rho_3 =$

$\max_{r\geqslant 1}\left\{\min\left\{2^{\alpha-1}\left(1+\frac{1}{r^\alpha}\right), 2^{\alpha-1}\phi^\alpha\left[1-\frac{\alpha r^{\alpha-1}}{(r+1)^\alpha}\right]\right\}\right\}$ for different values of α, we get that ρ_1 is better for $1 < \alpha \leqslant 1.44$, ρ_2 is better for $1.44 < \alpha < 2$ and ρ_3 is better for $\alpha \geqslant 2$. For different values of α, you can see the resulted ratios in Table 4.2.

Table 4.2: The values of the ratios given by the three analyses in relation with the value of α. In bold you can see the minimum value for each case.

α	1.25	1.5	1.75	2	2.25	2.5	2.75	3
ρ_1	**2.17**	2.91	3.90	5.23	7.02	9.41	12.63	16.94
ρ_2	2.37	**2.82**	**3.36**	4	4.75	5.65	**6.72**	**8**
ρ_3	0	0	0	**2.76**	**3.70**	**5.25**	**6.72**	**8**

Common Release, Power of 2 Deadlines

In this section, we consider that all jobs are released at time zero, but they have a different deadline. We assume that the deadlines are powers of 2 and that 2^k is the biggest deadline of our instance.

We present here CRP2D (Algorithm 6), an approximation algorithm with respect to energy. We split again the set of jobs $\mathcal{J}$ into two subsets: $A = \{j \in \mathcal{J} : t_j > \frac{u_j}{\phi}\}$ and $B = \{j \in \mathcal{J} : t_j \leqslant \frac{u_j}{\phi}\}$. We further split B into the subsets $B_\ell = \{j \in B : d_j = 2^\ell\}$, $0 \leqslant \ell \leqslant k$, with respect to the deadline of the jobs. As in the previous section, for each job in our instance, Algorithm 6 creates one or two jobs of an instance of the classical speed scaling setting. In order to analyze our algorithm, we define the three following instances of the classical speed scaling setting:

- I^*: $(0, d_j, \omega_j^*)\forall j \in \mathcal{J} = A \cup B$
- I': $(0, d_j, t_j)$ and $(0, d_j, w_j)\forall j \in B$ and $(0, d_j, u_j)\forall j \in A$
- $I'_{1/2}$: $(0, \frac{d_j}{2}, t_j)$ and $(\frac{d_j}{2}, d_j, w_j)\forall j \in B$ and $(0, d_j, u_j)\forall j \in A$

▶ **Lemma 4.10.** Let E^* and E' be the energy consumption in an optimal schedule for the instance I^* and I' respectively. Then, $E' \leqslant \phi^\alpha E^*$. ◀

Proof. Given an optimal solution for the instance I^*, we create a feasible schedule, $\mathcal{S}$, for the instance I'.

Consider an arbitrary job $j \in \mathcal{J}$ and its corresponding job $(0, d_j, \omega_j^*)$ of the instance I^* which is executed in q intervals in the optimal schedule for this instance: $(t_1, t'_1]$, $(t_2, t'_2]$,..., $(t_q, t'_q]$. Let s_p, $1 \leqslant p \leqslant q$, be the speed used in the interval $(t_p, t'_p]$. By

Algorithm 6: Common Release, Power of 2 Deadlines (CRP2D)

for *each job* $j \in \mathcal{J}$ **do**
  if $j \in B$, *i.e.,* $t_j \leqslant \frac{u_j}{\phi}$ **then**
    Add $(0, \frac{d_j}{2}, t_j)$ in set $\mathcal{Q}$;
  if $j \in A$, *i.e.,* $t_j > \frac{u_j}{\phi}$ **then**
    Add $(0, d_j, u_j)$ in set $\mathcal{W}$;
Run YDS algorithm to determine the speed $YDS^S(t)$ for each time $t \in (0, 2^k]$ for the jobs in $\mathcal{Q} \cup \mathcal{W}$;
In the interval $(0, \frac{1}{2}]$, execute the (parts of) jobs in $\mathcal{Q} \cup \mathcal{W}$ scheduled by YDS during this interval, using speed $s(t) = s^{YDS}(t)$;
for *each discrete time* $\frac{2^\ell}{2}$, $\ell = 0, 1, \ldots, k$ **do**
  // the queries for the jobs in B_ℓ are finished;
  for $j \in B_\ell$ **do**
    Add $(\frac{d_j}{2}, d_j, w_j)$ in set $\mathcal{W}^*_\ell$;
  In the interval $(\frac{2^\ell}{2}, 2^\ell]$, execute the (parts of) jobs in $\mathcal{Q} \cup \mathcal{W}$ scheduled by YDS during this interval as well as the jobs in $\mathcal{W}^*_\ell$, using speed $s(t) = s^{YDS}(t) + \sum_{j \in \mathcal{W}^*_\ell} \delta_j$;

definition, we have that

$$\omega^*_j = \sum_{p=1}^{q} \int_{t_p}^{t'_p} s_p dt = \sum_{p=1}^{q} (t'_p - t_p) s_p$$

In the schedule $\mathcal{S}$, we use in the interval $(t_p, t'_p]$, $1 \leqslant p \leqslant q$, the speed ϕs_p. Hence the work that can be executed in this interval is

$$\sum_{p=1}^{q} (t'_p - t_p)\phi s_p = \phi \sum_{p=1}^{q} (t'_p - t_p) s_p = \phi \omega^*_j \geqslant \omega_j$$

where the inequality follows from Lemma 4.1. Thus, in these intervals we can execute the jobs $(0, d_j, t_j)$ and $(0, d_j, w_j)$ or the job $(0, d_j, u_j)$ of the instance I'. By doing this for each job, we get a feasible schedule for the instance I' which at each time t uses speed ϕ times bigger than the speed of the optimal schedule for the instance I^*, and hence the energy consumption $E(\mathcal{S})$ in the schedule $\mathcal{S}$ is at most ϕE^*. Therefore, an optimal

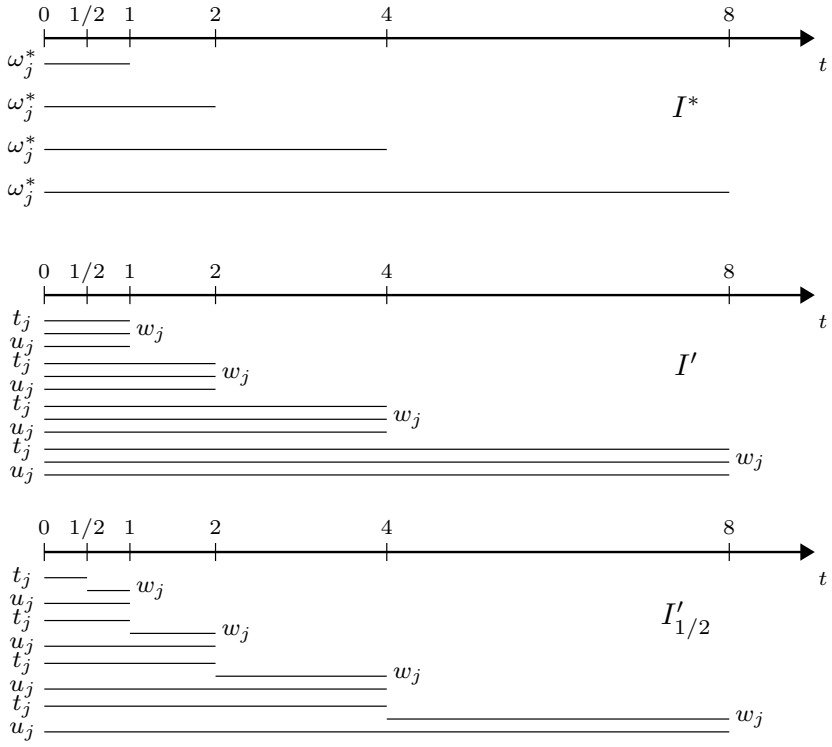


Figure 4.1: Intervals of the three different instances, I^*, I' and $I'_{1/2}$. On the top, there are the intervals of instance I^*, in the middle, the intervals of instance I' and on the bottom, the intervals of the instance $I'_{1/2}$. Note that the figure shows all possible intervals that can exist.

schedule for I' will use even smaller energy, i.e., $E' \leqslant E(\mathcal{S}) \leqslant \phi^\alpha E^*$, and the lemma follows. ■

▶ **Lemma 4.11.** Let E' and $E'_{1/2}$ be the energy consumption in an optimal schedule for the instance I' and $I'_{1/2}$ respectively. Then $E'_{1/2} \leqslant 2^\alpha E'$. ◀

Proof. We consider that both optimal solutions for the instances I' and $I'_{1/2}$ are created using the YDS algorithm. Let $s^*_{I'}(t)$ be the speed at each time t in an optimal schedule for the instance I'. Due to the YDS algorithm and the fact that the jobs have a common release date, this speed is non-increasing with respect to the time, i.e., $s^*_{I'}(t_1) \geqslant s^*_{I'}(t_2)$ for each $t_1 < t_2$. Moreover, the speed can change only at a deadline.

Note that the optimal schedule for I' is not feasible for $I'_{1/2}$. In order to make it feasible, we first transform the optimal schedule for I' into an intermediate schedule $\mathcal{S}$ which at each time t uses speed $s(t) = 2s^*_{I'}(t)$. Specifically, for any ℓ, $0 \leqslant \ell \leqslant k$, consider the work executed during the time interval $(2^{\ell-1}, 2^\ell]$. By doubling the speed during this interval, we can execute all this work during the first half, i.e., $(2^{\ell-1}, 2^\ell - 2^{\ell-2}]$, while the second half, i.e., $(2^\ell - 2^{\ell-2}, 2^\ell]$, remains idle. In a similar way we double the speed during $(0, \frac{1}{2}]$, and we are able to execute all of its work during $(0, \frac{1}{4}]$ while $(\frac{1}{4}, \frac{1}{2}]$ remains idle. By slightly abusing the definitions, we assume that the speed of the machine for any time t satisfies $s(t) = 2s^*_{I'}(t)$, even during the idle intervals of $\mathcal{S}$ where no work is executed. Note that, for each time interval $(0, 2^\ell]$, $-1 \leqslant \ell \leqslant k$, the half of it is idle in the constructed schedule $\mathcal{S}$. However, $\mathcal{S}$ is still not feasible for the instance $I'_{1/2}$. In what follows, we make $\mathcal{S}$ feasible by shifting some jobs in time.

For each job $j \in A$, there is a job $(0, d_j, u_j)$ which is added both in $I'_{1/2}$ and in I'. For these jobs, their allocation in $\mathcal{S}$ is already feasible since they have the same active interval in I' and $I'_{1/2}$.

For each job $j \in B_\ell$, $0 \leqslant \ell \leqslant k$, instance I' contains two jobs $(0, d_j, t_j)$ and $(0, d_j, w_j)$, while the instance $I'_{1/2}$ contains the jobs $(0, \frac{d_j}{2}, t_j)$ and $(\frac{d_j}{2}, d_j, w_j)$. Hence, in order to guarantee the feasibility of $\mathcal{S}$, we shift in $(\frac{d_j}{2}, d_j]$ the (parts of) jobs $(\frac{d_j}{2}, d_j, w_j)$ of $I'_{1/2}$ allocated in $(0, \frac{d_j}{2}]$. Similarly, we shift in $(0, \frac{d_j}{2}]$ the (parts of) jobs $(0, \frac{d_j}{2}, t_j)$ of $I'_{1/2}$ allocated in $(\frac{d_j}{2}, d_j]$. We make these shifts starting with the jobs having deadline 2^0, we continue with those having deadline 2^1, and so on.

We will prove by induction the following statement: "For each ℓ, $0 \leqslant \ell \leqslant k$, in the ℓ-th iteration of our shifting procedure, there is enough idle space in order to allocate all jobs $(0, 2^{\ell-1}, t_j)$ of $I'_{1/2}$ to the interval $(0, 2^{\ell-1}]$ and all jobs $(2^{\ell-1}, 2^\ell, w_j)$ of $I'_{1/2}$ to the interval $(2^{\ell-1}, 2^\ell]$".

- Basis: As explained before, the intervals $(\frac{1}{4}, \frac{1}{2}]$ and $(\frac{3}{4}, 1]$ in the schedule $\mathcal{S}$ are idle before any shifting due to the doubling of the speed. At the same time, the (parts of the) jobs $(0, \frac{1}{2}, t_j)$ which were infeasibly allocated after the doubling in $\mathcal{S}$ appear only in the interval $(\frac{1}{2}, \frac{3}{4}]$. Similarly, the (parts of the) jobs $(\frac{1}{2}, 1, w_j)$ which were infeasibly allocated after the doubling appear only in the interval $(0, \frac{1}{4}]$. Moreover, the speed during the whole interval $(0, 1]$ is constant, due to the YDS algorithm. Hence we can shift all the infeasible (parts of) jobs $(0, \frac{1}{2}, t_j)$ to the interval $(\frac{1}{4}, \frac{1}{2}]$ and all the infeasible (parts of) jobs $(\frac{1}{2}, 1, w_j)$ to the interval $(\frac{3}{4}, 1]$.
- Induction: Assume now that the statement is true for $\ell - 1$.

  Consider first the jobs $(0, 2^{\ell-1}, t_j)$ of $I'_{1/2}$. Some parts of these jobs may have been allocated in the interval $(2^{\ell-1}, 2^\ell - 2^{\ell-2}]$, making their execution infeasible. However, these parts are executed for at most $2^{\ell-2}$ time which corresponds exactly to the idle time during the interval $(0, 2^{\ell-1}]$. Thus, we can safely shift their execution to the left,

since the speed used in $(0, 2^{\ell-1}]$ is at least the speed used in $(2^{\ell-1}, 2^{\ell} - 2^{\ell-2}]$, by the definition of the YDS algorithm, getting a feasible schedule for these jobs.

Consider now the the jobs $(2^{\ell-1}, 2^{\ell}, w_j)$ of $I'_{1/2}$. Some parts of these jobs may have been allocated in the interval $(0, 2^{\ell-1}]$, making their execution infeasible. However, these parts are executed for at most $\frac{1}{4} + 2^{-2} + 2^{-1} + \cdots + 2^{\ell-3} = 2^{\ell-2}$ time which corresponds exactly to the idle time during the interval $(2^{\ell} - 2^{\ell-2}, 2^{\ell}]$. If the speed used in $(0, 2^{\ell-1}]$ is equal to the speed used in $(2^{\ell-1}, 2^{\ell}]$, then we can safely shift their execution to the right, getting a feasible schedule for these jobs. If the speed used in $(0, 2^{\ell-1}]$ is bigger than the speed used in $(2^{\ell-1}, 2^{\ell}]$, then the jobs $(0, 2^{\ell}, w_j)$ of I' are not executed at all during $(2^{\ell-1}, 2^{\ell}]$ in the optimal schedule for the instance I' obtained by the YDS algorithm, since they belong on a different critical interval. Hence, the jobs $(2^{\ell-1}, 2^{\ell}, w_j)$ of $I'_{1/2}$ are also not executed in $(2^{\ell-1}, 2^{\ell}]$. They are already feasible.

As a result, the schedule $\mathcal{S}$ is feasible for $I'_{1/2}$ after all the shifts, and it uses speed $s(t) = 2s^*_{I'}(t)$, for any time t. Then, the energy consumption $E(\mathcal{S})$ of $\mathcal{S}$ is at most two time the energy consumption of the optimal solution for I'. Therefore, an optimal schedule for $I'_{1/2}$ will use even smaller energy, i.e., $E'_{1/2} \leqslant E(\mathcal{S}) \leqslant 2^{\alpha} E'$, and the lemma follows. ■

▶ **Lemma 4.12.** Given an optimal schedule for the instance $I'_{1/2}$ and a schedule given by Algorithm 6, we have that $s(t) \leqslant 2s^*_{I'_{1/2}}(t)$ for each time instant t. ◀

Proof. By the construction of $I'_{1/2}$ and the definition of Q and $\mathcal{W}$ in Lines 1-5 of the algorithm, we have that $Q \cup \mathcal{W} \subseteq I'_{1/2}$. In Line 6, an optimal schedule is created for the jobs in $Q \cup \mathcal{W}$. Since both optimal solutions for $I'_{1/2}$ and for the jobs in $Q \cup \mathcal{W}$ are computed by the YDS algorithm, and due to the properties of this algorithm, we have that $s^{YDS}(t) \leqslant s^*_{I'_{1/2}}(t)$, for each $t \in (0, 2^k]$.

Similarly, by the construction of $I'_{1/2}$ and the definition of $\mathcal{W}^*_{\ell}$'s in Lines 8-11 of the algorithm, we have that $\bigcup_{\ell=0}^{k} \mathcal{W}^*_{\ell} \subseteq I'_{1/2}$. Moreover, the jobs in $\mathcal{W}^*_{\ell}$, $0 \leqslant \ell \leqslant k$, are of the form $(\frac{2^{\ell}}{2}, 2^{\ell}, w_j)$, and hence the active intervals of any two jobs belonging to two different sets $\mathcal{W}^*_{\ell}$ and $\mathcal{W}^*_{\ell'}$ are time-disjoint. Thus, in an optimal solution for the jobs in $\bigcup_{\ell=0}^{k} \mathcal{W}^*_{\ell}$, the speed used during the interval $(\frac{2^{\ell}}{2}, 2^{\ell}]$ is $\sum_{j \in \mathcal{W}^*_{\ell}} \delta_j$. Therefore, using the same arguments as before, for each $t \in (\frac{2^{\ell}}{2}, 2^{\ell}]$, we have that $\sum_{j \in \mathcal{W}^*_{\ell}} \delta_j \leqslant s^*_{I'_{1/2}}(t)$.

For the speed of the algorithm, for any time $t \in (0, \frac{1}{2}]$, we have that $s(t) = s^{YDS}(t) \leqslant s^*_{I'_{1/2}}$ (see Line 7). Moreover, for any ℓ, $0 \leqslant \ell \leqslant k$, and any time $t \in (\frac{2^{\ell}}{2}, 2^{\ell}]$, we have that $s(t) = s^{YDS}(t) + \sum_{j \in \mathcal{W}^*_{\ell}} \delta_j \leqslant s^*_{I'_{1/2}}(t) + s^*_{I'_{1/2}}(t) \leqslant 2s^*_{I'_{1/2}}(t)$ (see Line 12), and the lemma follows. ■

Algorithm 7: Common Release, Arbitrary Deadlines (CRAD)

```
for each job j ∈ 𝒥 do
    d'_j = max_i{2^i | 2^i ⩽ d_j};
    Add job (r_j, d'_j, t_j, u_j, w_j) in instance Ǐ;
Run Algorithm 6 with the updated instance Ǐ;
```

▶ **Corollary 4.13.** Let E and $E'_{1/2}$ be the energy consumption of the schedule created by Algorithm 6 and of an optimal schedule for the instance $I'_{1/2}$ respectively. Then, $E \leqslant 2^{\alpha} E'_{1/2}$. ◀

▶ **Theorem 4.14.** CRP2D (Algorithm 6) achieves a competitive ratio of $(4\phi)^{\alpha}$ with respect to energy. ◀

Proof. Note that the energy consumption of an optimal schedule for our original instance and of an optimal schedule for the instance I^* is exactly the same, as they contain exactly the same set of jobs with the same characteristics. Then, the proof of the theorem is an immediate consequence of Lemmas 4.10 and 4.11, and Corollary 4.13. Specifically, we have: $E \leqslant 2^{\alpha} E'_{1/2} \leqslant 4^{\alpha} E' \leqslant (4\phi)^{\alpha} E^*$. ■

Common Release, Arbitrary Deadlines

In this section we adapt the previous result to jobs with arbitrary deadlines. Given an instance I of our original problem, we create an instance $\check{I}$ by rounding down the deadline of all jobs to a power of two: for each job $(r_j, d_j, t_j, u_j, w_j) \in \mathcal{J}$, add a job $(r_j, d'_j, t_j, u_j, w_j)$ in $\check{I}$, where $d'_j = \max\{2^i | 2^i \leqslant d_j\}$. Then, run Algorithm 6 using instance $\check{I}$ as input. We call this algorithm CRAD (Algorithm 7).

▶ **Lemma 4.15.** Let E and $\check{E}$ be the energy consumption of an optimal schedule for the instance I and $\check{I}$ respectively. Then, $\check{E} \leqslant 2^{\alpha} E$. ◀

Proof. Let $\mathcal{S}$ be the optimal schedule for I. We create a schedule $\mathcal{S}'$ by doubling the speed of $\mathcal{S}$ at each time t. Then we shift the work of all jobs as early in time as possible such that to use this bigger speed. Note that the order of execution of the (parts of) jobs does change. This schedule uses twice the speed of $\mathcal{S}$ so for its energy consumption $E(\mathcal{S}')$ we have $E(\mathcal{S}') \leqslant 2^{\alpha} E$. We will next show that the schedule $\mathcal{S}'$ is feasible for the instance $\check{I}$.

Let $s(t)$ be the speed at time t in the schedule $\mathcal{S}$. Note that the schedule $\mathcal{S}$ is an optimal schedule using the values ω^*_j, and it can be constructed by the YDS algorithm. Since, all jobs have a common release time, we know that $s(t)$ is non-increasing with

respect to t, as mentioned before (see proof of Lemma 4.11). Consider now any time t and let W_t be the total work executed in the interval $(0, t]$ in $\mathcal{S}$, that is $W_t = \int_0^t s(t)dt$. Since the speeds in $\mathcal{S}$ are non-increasing, the execution of the work W_t finishes the latest at time $\frac{t}{2}$ in $\mathcal{S}'$. Hence, for the completion time C'_j of each job j in $\mathcal{S}'$ we have that $C'_j \leqslant \frac{C_j}{2}$, where C_j is the completion time of j in $\mathcal{S}$. By definition, $d'_j \geqslant \frac{d_j}{2}$. Then, $C'_j \leqslant \frac{C_j}{2} \leqslant \frac{d_j}{2} \leqslant d'_j$. Therefore, the job j is feasibly executed in $\mathcal{S}'$ and $\mathcal{S}'$ is feasible for the instance $\check{I}$.

Since the energy $\check{E}$ of an optimal schedule for the instance $\check{I}$ is smaller that the energy of any feasible schedule, we have that $\check{E} \leqslant E(\mathcal{S}')$ and the lemma follows. ■

▶ **Corollary 4.16.** CRAD (Algorithm 7) achieves a competitive ratio of $(8\phi)^\alpha$ with respect to energy. ◀

4.3.3 Online Model

In this section, we consider the QBSS model when the jobs arrive online and they should be executed on a single machine.

AVR with Queries

The online AVR algorithm for the classical speed scaling setting works as follows: at each time t, the machine runs at speed $s^{AVR}(t) = \sum_{j:t\in(r_j,d_j]} \delta_j$ and it executes the unfinished job with the smaller deadline which is released before t. Yao et al. [YDS95] proved that AVR is $2^{\alpha-1}\alpha^\alpha$-competitive with respect to energy.

In this section we propose the online algorithm AVRQ (Algorithm 8), an adaptation of AVR to the QBSS model. AVRQ does the query for all the jobs by selecting as a splitting point the half of their interval. Specifically, for each job $(r_j, d_j, t_j, u_j, w_j)$ in $\mathcal{J}$, two jobs of the classical speed scaling setting are created and added to the set $\mathcal{J}'$ (in an online manner): the job $(r_j, \frac{r_j+d_j}{2}, t_j)$ at time r_j, and the job $(\frac{r_j+d_j}{2}, d_j, w_j)$ at time $\frac{r_j+d_j}{2}$. The AVR algorithm runs using as input the set of jobs $\mathcal{J}'$ which is created online. The following lemma extends the lower bound for AVR proposed in [BKP07] and gives a lower bound to the competitive ratio of AVRQ with respect to energy.

▶ **Lemma 4.17.** The competitive ratio of algorithm AVRQ is at least $(2\alpha)^\alpha$ with respect to energy. ◀

Proof. Consider an instance where all the jobs have the same deadline n. A job, j_i, arrives at time $i = 0, 1, \ldots, n-1$, having $(r_j, d_j, t_j, u_j, w_j) = (i, n, h_i^{1/\alpha}, (h_i+\epsilon)^{1/\alpha}, (h_i+\epsilon)^{1/\alpha})$, where $h_i = \frac{1}{n-i}$.

If ϵ is small enough, the optimal energy algorithm does not make the query and completes the job that arrives at time i by time $i+1$ running at speed $\left(\frac{1}{n-i}+\epsilon\right)^{1/\alpha}$.

Algorithm 8: AVRQ

```
for each time instant t do
    for each job j ∈ 𝒥 do
        if t == r_j then
            Add job (r_j, (r_j+d_j)/2, t_j) in instance I';
        else if t == (r_j+d_j)/2 then
            Add job ((r_j+d_j)/2, d_j, w_j) in instance I';
    Run AVR with the updated instance I';
```

Following the same calculations as in Bansal et al. paper [BKP07], the resulting energy is $H_n + O(1)$.

Now, let us try to analyze the energy usage of AVRQ. In the interval $[0, n/2]$ only queries are executed and the energy consumption is about $\ln 2$. Since these values don't make a significant contribution to the total energy consumption, we discard it. Let us estimate the energy consumption in the interval $[n/2, n]$. Suppose that n is even.

Let $I_i = \left[\frac{n+i}{2}, \frac{n+i+1}{2}\right]$ for $i = 0, \ldots, n-1$. If i is even, the queries which correspond to jobs $j_{i+1}, \ldots, j_{\frac{n+1}{2}}$ are executed in the interval I_i. If i is odd, the queries which correspond to jobs $j_{i+1}, \ldots, j_{\frac{n+1-1}{2}}$ are executed in the interval I_i. For every i, the jobs $j_0, \ldots, j_i$ are executed in the interval I_i. Calculating the speed for each interval, we get

$$s(i) = 2\sum_{j=0}^{i} \frac{\left(h_j + \epsilon\right)^{1/\alpha}}{n-j} + 2\sum_{j=i+1}^{\frac{n+i}{2}} \frac{h_j^{1/\alpha}}{n-j}, \text{ when i is even}$$

$$s(i) = 2\sum_{j=0}^{i} \frac{\left(h_j + \epsilon\right)^{1/\alpha}}{n-j} + 2\sum_{j=i+1}^{\frac{n+i-1}{2}} \frac{h_j^{1/\alpha}}{n-j}, \text{ when i is odd}$$

In total, for any i, we have

$$s(i) \geqslant 2\sum_{j=0}^{\frac{n+i-1}{2}} \frac{h_j^{1/\alpha}}{n-j} \geqslant 2\sum_{j=0}^{\frac{n+i-1}{2}} \frac{1}{(n-j)^{1+\frac{1}{\alpha}}} \geqslant 2\int_0^{\frac{n+i-3}{2}} \frac{1}{(n-j)^{1+\frac{1}{\alpha}}}\, dj$$

$$= \left[2\alpha(n-j)^{-1/\alpha}\right]_0^{\frac{n+i-3}{2}} = 2\alpha\left(\frac{2n-n-i+3}{2}\right)^{-1/\alpha} - 2\alpha n^{-1/\alpha}$$

$$= 2\alpha 2^{1/\alpha}\left[(n-i+3)^{-1/\alpha} - (2n)^{-1/\alpha}\right] \tag{4.1}$$

The total energy consumption of AVRQ is

$$\begin{aligned} E &= \frac{1}{2}\sum_{i=0}^{n-1} s(i)^\alpha \geqslant \frac{1}{2}(2\alpha)^\alpha 2 \sum_{i=0}^{n-1}\left((n-i+3)^{-1/\alpha} - (2n)^{-1/\alpha}\right)^\alpha \\ &\geqslant (2\alpha)^\alpha \left(\sum_{i=0}^{n-1}\frac{1}{n-i+3} - \alpha\sum_{i=0}^{n-1}\left(\frac{1}{n-i+3}\right)^{\frac{\alpha-1}{\alpha}}(2n)^{-1/\alpha}\right) \\ &\geqslant (2\alpha)^\alpha\left(H_n - 2 - \alpha^2(2n)^{-1/\alpha}\,\Theta(n^{1/\alpha})\right) \\ &= (2\alpha)^\alpha(H_n - \Theta(1)) \end{aligned}$$

The first inequality is given by 4.1. For the second inequality we use the property $(x-y)^\alpha \geqslant x^\alpha - ax^{\alpha-1}y$ with $x = (n-i+3)^{-1/\alpha}$ and $y = (2n)^{-1/\alpha}$, which is proved in [BKP07] for $\alpha > 1$. So, if n is large enough, the competitive ratio can be made arbitrarily close to $(2\alpha)^\alpha$. ■

Let AVR* be the original AVR algorithm when executed using the set of jobs $\mathcal{J}^*$ created as follows: for each $j \in \mathcal{J}$, add the job (r_j, d_j, ω_j^*) to $\mathcal{J}^*$. The following theorem compares, for each time t, the speed used by the algorithm AVRQ with the speed of AVR*.

▶ **Theorem 4.18.** For any time instant t, we have $s^{AVRQ}(t) \leqslant 2s^{\text{AVR}^*}(t)$. ◀

Proof. At any time t, the speed of AVRQ is

$$\begin{aligned} s^{AVRQ}(t) &\leqslant \textstyle\sum_{j\in\mathcal{J}:t\in(r_j,d_j]} \max\left\{\frac{t_j}{(d_j-r_j)/2}, \frac{w_j}{(d_j-r_j)/2}\right\} = 2\sum_{j\in\mathcal{J}:t\in(r_j,d_j]}\frac{\max\{t_j,w_j\}}{d_j-r_j} \\ &\leqslant 2\sum_{j\in\mathcal{J}:t\in(r_j,d_j]}\frac{\min\{u_j,t_j+w_j\}}{d_j-r_j} = 2\sum_{j\in\mathcal{J}^*:t\in(r_j,d_j]}\frac{\omega_j^*}{d_j-r_j} \\ &= 2s^{\text{AVR}^*}(t) \end{aligned}$$

■

▶ **Corollary 4.19.** AVRQ (Algorithm 8) is $2^{2\alpha-1}\alpha^\alpha$-competitive with respect to energy. ◀

BKP with Queries

The online BKP algorithm for the classical speed scaling setting works as follows: for the time instants t, t_1 and t_2 with $t_1 < t \leqslant t_2$, let $w(t,t_1,t_2)$ be the total work of jobs that have arrived by time t, have a release time of at least t_1 and a deadline of at most t_2. At any time t, the machine runs at speed $s^{BKP}(t) = e\max_{t_1,t_2}\frac{w(t,t_1,t_2)}{(t_2-t_1)}$ and it executes the

Algorithm 9: BKPQ

```
for each time instant t do
    for each job j ∈ 𝒥 do
        if t == r_j then
            if t_j ⩽ u_j/φ then
                Add job (r_j, (r_j+d_j)/2, t_j) in instance I′;
            else if t_j > u_j/φ then
                Add job (r_j, d_j, u_j) in instance I′;
        else if t == (r_j+d_j)/2 then
            if t_j ⩽ u_j/φ then
                Add job ((r_j+d_j)/2, d_j, w_j) in instance I′;
    Run BKP with the updated instance I′;
```

unfinished job with the smallest deadline which is released before t. Bansal et al. proved that BKP achieves a competitive ratio of $2(\frac{\alpha}{\alpha-1})^\alpha e^\alpha$ with respect to energy, while it is e-competitive with respect to maximum speed.

In this section, we propose the online algorithm BKPQ (Algorithm 9), an adaptation of BKP to the QBSS model. For each job $(r_j, d_j, t_j, u_j, w_j)$ in $\mathcal{J}$, BKPQ decides to make the query only if $t_j \leqslant \frac{u_j}{\phi}$ using as splitting point $\tau_j = \frac{r_j+d_j}{2}$. Hence, in the case of a query, two jobs of the classical speed scaling setting, corresponding to $(r_j, d_j, t_j, u_j, w_j)$, are created and added to the set of jobs $\mathcal{J}'$ (in an online manner): the job $(r_j, \frac{r_j+d_j}{2}, t_j)$ at time r_j, and the job $(\frac{r_j+d_j}{2}, d_j, w_j)$ at time $\frac{r_j+d_j}{2}$. In the case where no query is made, then a single job, corresponding to $(r_j, d_j, t_j, u_j, w_j)$, is added to the set $\mathcal{J}'$: the job (r_j, d_j, u_j) at time r_j. The BKP algorithm runs using as input the set of jobs $\mathcal{J}'$ which is created online.

Let BKP* be the original BKP algorithm when executed using the set of jobs $\mathcal{J}^*$ created as follows: for each $j \in \mathcal{J}$, add the job (r_j, d_j, ω_j^*) to $\mathcal{J}^*$. The following theorem compares, for each time t, the speed used by the algorithm BKPQ with the speed of BKP*.

▶ **Theorem 4.20.** For any time instant t, we have $s^{BKPQ}(t) \leqslant (2+\phi)s^{\text{BKP}^*}(t)$. ◀

Proof. Let t_1 and t_2 be time instants such that

$$\frac{w(t,t_1,t_2)}{(t_2-t_1)} = \max_{t_1',t_2'} \frac{w(t,t_1',t_2')}{(t_2'-t_1')}$$

for the jobs in $\mathcal{J}'$. We define three disjoint subsets of $\mathcal{J}$ and we explain how the corresponding jobs in $\mathcal{J}'$ contribute to $w(t,t_1,t_2)$:

- Let $\mathcal{L}$ be the set of queried jobs whose queries are entirely processed in the interval $(t_1,t_2]$, but not its exact loads themselves, and start before time t.
- Let $\mathcal{R}$ be the set of queried jobs whose exact load is entirely processed in the interval $(t_1,t_2]$, but not its queries themselves, and start before time t.
- Let C be the set of jobs (corresponding to queries, exact loads or initial workloads) that are entirely processed in the interval $(t_1,t_2]$ and start before time t.

Let $W(\mathcal{L}) = \sum_{j\in\mathcal{L}} t_j$, $W(\mathcal{R}) = \sum_{j\in\mathcal{R}} w_j$ and $W(C) = \sum_{j\in C} \omega_j$ be the total work of jobs in $\mathcal{J}'$ which belong to $\mathcal{L}$, $\mathcal{R}$ and C, respectively. By definition, $W(\mathcal{L})+W(\mathcal{R})+W(C)$ describes the total work that is executed in $(t_1,t_2]$ by BKPQ. So, for any time $t\in(t_1,t_2]$, we have that $s^{BKPQ}(t) = \frac{W(\mathcal{L})+W(\mathcal{R})+W(C)}{(t_2-t_1)}$.

In a similar way, let $W^*(\mathcal{L}) = \sum_{j\in\mathcal{L}} \omega_j^*$, $W^*(\mathcal{R}) = \sum_{j\in\mathcal{R}} \omega_j^*$ and $W(C) = \sum_{j\in C} \omega_j^*$. We consider the following three bounds:

1. Consider a job $j\in\mathcal{L}$. If $\omega_j^* = u_j$ then $t_j \leqslant \frac{u_j}{\phi} \leqslant u_j = \omega_j^*$. If $\omega_j^* = t_j + w_j$ then $t_j \leqslant t_j + w_j = \omega_j^*$. Thus, for each $j\in\mathcal{L}$ we have $t_j \leqslant \omega_j^*$ and $W(\mathcal{L}) \leqslant W^*(\mathcal{L})$.
2. Consider a job $j\in\mathcal{R}$. If $\omega_j^* = u_j$ then $w_j \leqslant u_j = \omega_j^*$. If $\omega_j^* = t_j + w_j$ then $w_j \leqslant t_j + w_j = \omega_j^*$. Thus, for each $j\in\mathcal{R}$ we have $w_j \leqslant \omega_j^*$ and $W(\mathcal{R}) \leqslant W^*(\mathcal{R})$.
3. For each $j\in C$ we have $\omega_j \leqslant \phi\omega_j^*$ by using Lemma 4.1. Thus, $W(C) \leqslant \phi W^*(C)$.

Consider now the time interval (t_0,t_3) such as $t_0 = \max\{0, 2t_1 - t_2\}$ and $t_3 = 2t_2 - t_1$. For each $j\in\mathcal{L}$ we have $t_1 \leqslant r_j \leqslant t < \tau_j \leqslant t_2$, and hence $d_j = 2\tau_j - r_j \leqslant 2t_2 - t_1 = t_3$. For each $j\in\mathcal{R}$ we have $t_1 \leqslant \tau_j$ and $d_j \leqslant t_2$, and hence $r_j = 2\tau_j - d_j \geqslant \max\{0, 2t_1 - t_2\} = t_0$. Therefore, each job $j\in\mathcal{L}\cup\mathcal{R}\cup C$ should be executed in the interval $[t_0,t_3]$ by any algorithm and also by BKP*. As a result, we have:

$$s^{\text{BKP}^*}(t) \geqslant \frac{W^*(\mathcal{L})+W^*(\mathcal{R})+W^*(C)}{(t_3-t_0)} = \frac{W^*(\mathcal{L})+W^*(\mathcal{R})+W^*(C)}{3(t_2-t_1)} \tag{4.2}$$

As explained before, for the speed of BKPQ at any time $t\in(t_1,t_2]$ we have

$$\begin{aligned} s^{BKPQ}(t) = \frac{W(\mathcal{L})+W(\mathcal{R})+W(C)}{(t_2-t_1)} &\leqslant \frac{W^*(\mathcal{L})+W^*(\mathcal{R})+\phi W^*(C)}{(t_2-t_1)} \\ &= \frac{W^*(\mathcal{L})+W^*(\mathcal{R})+W^*(C)+(\phi-1)W^*(C)}{(t_2-t_1)} \end{aligned}$$

$$\leqslant 3s^{\text{BKP}^*}(t) + (\phi - 1)s^{\text{BKP}^*}(t) = (2 + \phi)s^{\text{BKP}^*}(t)$$

The first inequality follows by the three bounds presented above. In the next line we just add and subtract $W^*(C)$. For the last inequality we use Inequality 4.2, as well as the fact that $s^{\text{BKP}^*}(t) \geqslant \frac{W^*(C)}{(t_2 - t_1)}$ since all jobs in the set C are entirely executed in the interval $(t_1, t_2]$ by any algorithm and also by BKP*. ■

▶ **Corollary 4.21.** BKPQ (Algorithm 9) is $(2 + \phi)^\alpha 2(\frac{\alpha}{\alpha - 1})^\alpha e^\alpha$-competitive with respect to energy, and $(2 + \phi)e$-competitive with respect to maximum speed. ◀

4.4 Multiple Machines

In this section we adapt to the QBSS model the online AVR(m) algorithm proposed by Albers et al. [AAG15] for the classical speed scaling setting on a set of m parallel identical machines $\mathcal{M}$. AVR(m) is $(2^{\alpha-1}\alpha^\alpha + 1)$-competitive with respect to energy consumption.

For completeness, we briefly present AVR(m). The algorithm works online per each unit time slot $(t, t + 1]$ and it schedules δ_j amount of work from each active job during $(t, t + 1]$. Let $\mathcal{J}_t$ be the set of active jobs in $(t, t + 1]$. Moreover, let $U \subseteq \mathcal{J}_t$ be the unscheduled jobs of $\mathcal{J}_t$ and $R \subseteq \mathcal{M}$ be the remaining unused machines at each step of the algorithm. In the beginning, we set $U = \mathcal{J}_t$ and $R = \mathcal{M}$. We denote by $\Delta = \sum_{j \in U} \delta_j$ the total work of the jobs in U. The jobs in U will be characterized as *big* or *small* depending on their densities. Intuitively, each *big* job will occupy one machine during the whole slot $(t, t + 1]$, while *small* jobs will share the remaining machines in $(t, t + 1]$. The algorithm searches in an iterative way the job $\hat{\jmath} = \text{argmax}\{\delta_j : j \in U\}$ with the maximum density in U and if $\delta_{\hat{\jmath}} > \frac{\Delta}{|R|}$ then it is characterized as a *big* one. In this case, the algorithm schedules $\hat{\jmath}$ with speed $\delta_{\hat{\jmath}}$ in the machine of the lower index in R which is then removed from R. Moreover, the algorithm updates $\mathcal{J}_t = \mathcal{J}_t \setminus \{\hat{\jmath}\}$ and it searches for the next *big* job. If no big job exists, then all the remaining jobs are *small* and they are allocated to the remaining machines using speed $\frac{\Delta}{|R|}$. Note that, at each time moment the speed of a machine with lower index is not less than the speed of a machine with larger index.

Here, we propose the online algorithm AVRQ(m) (Algorithm 10) which makes the query for all jobs by selecting as a splitting point the half of their interval. Specifically, for each job $(r_j, d_j, t_j, u_j, w_j)$ in $\mathcal{J}$, two jobs of the classical speed scaling setting are created and added to the set $\mathcal{J}'$ (in an online manner): the job $\zeta(j) = (r_j, \frac{r_j + d_j}{2}, t_j)$ at time r_j, and the job $\zeta'(j) = (\frac{r_j + d_j}{2}, d_j, w_j)$ at time $\frac{r_j + d_j}{2}$. The AVR(m) algorithm runs using as input the set of jobs $\mathcal{J}'$ which is created online.

We start our analysis with two technical lemmas.

Algorithm 10: $AVRQ_{(m)}$

for *each time instant* t **do**
  for *each job* $j \in \mathcal{J}$ **do**
    if $t == r_j$ **then**
      Add job $(r_j, \frac{r_j+d_j}{2}, t_j)$ in instance I';
    else if $t == \frac{r_j+d_j}{2}$ **then**
      Add job $(\frac{r_j+d_j}{2}, d_j, w_j)$ in instance I';
  Run $AVR_{(m)}$ with the updated instance I';

▶ **Lemma 4.22.** Let two sets of non-negative rational numbers $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$ be given such that $b_j \leqslant 2a_j$ for all $j = 1, \ldots, n$. Let π_A and π_B be permutations of numbers from A and B, respectively, in which the numbers are ordered in non-increasing order. Then $\pi_B(i) \leqslant 2\pi_A(i)$ for all $i = 1, \ldots, n$. ◀

Proof. Let the elements of the set A and B be numbered as in the permutation π_A. Assume that $\pi_B(i) = b_k$ and $k \geqslant i$. We have $\pi_B(i) \leqslant 2\pi_A(k) \leqslant 2\pi_A(i)$. Let $k < i$. From pigeonhole principle, the exists $\pi_B(l) = b_j$ such that $l < i$ and $j \geqslant i$. We get $\pi_B(i) \leqslant \pi_B(l) = b_j \leqslant 2a_j \leqslant 2a_i = 2\pi_A(i)$. ■

▶ **Lemma 4.23.** Let a sequence of non-negative rational numbers $a_1, \ldots, a_n$ and an integer $m \geqslant 2$ be given. If $a_1 > \frac{\sum_{i=1}^n a_i}{m}$, then $\frac{\sum_{i=1}^n a_i}{m} > \frac{\sum_{i=2}^n a_i}{m-1}$, otherwise, $\frac{\sum_{i=1}^n a_i}{m} \leqslant \frac{\sum_{i=2}^n a_i}{m-1}$. ◀

Proof. We have:

$$\frac{\sum_{i=1}^n a_i}{m} = \frac{(\sum_{i=2}^n a_i + a_1)(m-1)}{m(m-1)} = \frac{\sum_{i=2}^n a_i}{m-1} + \frac{a_1 m - \sum_{i=1}^n a_i}{m(m-1)}$$

The last term is positive when $a_1 > \frac{\sum_{i=1}^n a_i}{m}$, and non-positive otherwise, hence the assertion of the lemma follows. ■

Let AVR$^*(m)$ be the original AVR(m) algorithm when executed using the set of jobs $\mathcal{J}^*$ created as follows: for each $j \in \mathcal{J}$, add the job (r_j, d_j, ω_j^*) to $\mathcal{J}^*$. The following theorem compares, for each time t, the speed used by the algorithm AVRQ(m) with the speed of AVR$^*(m)$.

▶ **Theorem 4.24.** For any time instant t, and any machine i, we have $s_i^{\text{AVRQ}(m)}(t) \leqslant 2s_i^{\text{AVR}^*(m)}(t)$. ◀

Proof. We consider the set of jobs $\mathcal{J}''$ which is produced from $\mathcal{J}^*$ by replacing each job (r_j, d_j, ω_j^*) with two jobs,
$\psi(t) = (r_j, \frac{r_j+d_j}{2}, \frac{\omega_j^*}{2})$ and $\psi'(t) = (\frac{r_j+d_j}{2}, d_j, \frac{\omega_j^*}{2})$. Since the number of jobs and their densities in each unit time slot $(t, t+1]$ do not change, then the speed of the machines in the schedules obtained by AVR(m) when applied to the sets of jobs $\mathcal{J}^*$ and $\mathcal{J}''$ does not change either.

Algorithm AVRQ(m) also creates two jobs, let $\zeta(j)$ and $\zeta'(j)$ for each original job $j \in \mathcal{J}$ using the same intervals as in $\mathcal{J}''$. Hence, the number of active jobs in $\mathcal{J}'$ and $\mathcal{J}''$ is the same at each unit time slot, and by definition we have

$$\delta_{\zeta(j)} \leqslant 2\delta_{\psi(j)} \text{ and } \delta_{\zeta'(j)} \leqslant 2\delta_{\psi'(j)} \text{ for all } j \in \mathcal{J} \tag{4.3}$$

since $t_j \leqslant \omega_j^*$ and $w_j \leqslant \omega_j^*$.

Denote by $\mathcal{J}_t'' \subseteq \mathcal{J}''$ and $\mathcal{J}_t' \subseteq \mathcal{J}'$ the set of active jobs in the unit slot $(t, t+1]$. We order the jobs in each set in non-increasing densities. Note that $|\mathcal{J}_t''| = |\mathcal{J}_t'| = r$. Let a_j be the density of the j-th job in $\mathcal{J}''$ and b_j be the density of the j-th job in $\mathcal{J}'$. Lemma 4.22 and Inequalities (4.3) imply that $b_j \leqslant 2a_j$.

Let k be the number of *big* jobs in the set $\mathcal{J}_t''$ and ℓ be the number of *big* jobs in the set $\mathcal{J}_t'$. We consider three cases.

1. Case $k = \ell$. For each machine $i \leqslant k$ we have:

$$s_i^{\text{AVRQ}(m)}(t) = b_i \leqslant 2a_i = 2s_i^{\text{AVR}^*(m)}(t)$$

For each machine $i > k$ we have:

$$s_i^{\text{AVRQ}(m)}(t) = \frac{\sum_{j=\ell+1}^{r} b_j}{m-\ell} \leqslant 2\frac{\sum_{j=k+1}^{r} a_j}{m-k} \leqslant 2s_i^{\text{AVR}^*(m)}(t)$$

2. Case $k > \ell$. For each machine $i \leqslant \ell$ we have:

$$s_i^{\text{AVRQ}(m)}(t) = b_i \leqslant 2a_i = 2s_i^{\text{AVR}^*(m)}(t)$$

For each machine $i > \ell$ we have $b_h \leqslant \frac{\sum_{j=h}^{r} b_j}{m-h+1}$ for $\ell+1 \leqslant h \leqslant k$. Successively applying Lemma 4.23 we get:

$$\frac{\sum_{j=\ell+1}^{r} b_j}{m-\ell} \leqslant \frac{\sum_{j=\ell+2}^{r} b_j}{m-\ell-1} \leqslant \cdots \leqslant \frac{\sum_{j=k+1}^{r} b_j}{m-k}$$

Hence, we obtain:

$$s_i^{\text{AVRQ}(m)}(t) = \frac{\sum_{j=\ell+1}^{r} b_j}{m-\ell} \leqslant \frac{\sum_{j=k+1}^{r} b_j}{m-k} \leqslant 2\frac{\sum_{j=k+1}^{r} a_j}{m-k} \leqslant 2s_i^{\text{AVR}^*(m)}(t)$$

3. Case $k < \ell$. For each machine $i \leqslant k$ we have

$$s_i^{\mathrm{AVRQ}(m)}(t) = b_i \leqslant 2a_i = 2s_i^{\mathrm{AVR}^*(m)}(t)$$

For each machine i, $k < i \leqslant \ell$, we have

$$s_i^{\mathrm{AVRQ}(m)}(t) = b_i \leqslant 2a_i \leqslant 2\frac{\sum_{j=k+1}^{r} a_j}{m-k} = 2s_i^{\mathrm{AVR}^*(m)}(t)$$

where the last inequality follows by the definition of $\mathrm{AVR}^*(m)$. For each machine $i > \ell$ we have $b_h > \frac{\sum_{j=h}^{r} b_j}{m-h+1}$ for $k+1 \leqslant h \leqslant \ell$. Successively applying Lemma 4.23 we get:

$$\frac{\sum_{j=k+1}^{r} b_j}{m-k} > \frac{\sum_{j=\ell+2}^{r} b_j}{m-k-1} > \cdots > \frac{\sum_{j=\ell+1}^{r} b_j}{m-\ell}$$

Hence, we obtain:

$$s_i^{\mathrm{AVRQ}(m)}(t) \leqslant \frac{\sum_{j=\ell+1}^{r} b_j}{m-\ell} < \frac{\sum_{j=k+1}^{r} b_j}{m-k} \leqslant 2\frac{\sum_{j=k+1}^{r} a_j}{m-k} \leqslant 2s_i^{\mathrm{AVR}^*(m)}(t)$$

and the theorem follows. ■

▶ **Corollary 4.25.** AVRQ(m) (Algorithm 10) is $2^{\alpha}(2^{\alpha-1}\alpha^{\alpha}+1)$-competitive with respect to energy. ◀

4.5 Conclusion

In this chapter, we studied an enhanced speed scaling setting, where queries can be additionally executed in the system in order to reveal a more accurate value of the workload of jobs. This model in particular makes sense in the context where an operation (query) can decrease the length of a task (as happens in the case of code optimization or file compression). The main objective was the minimization of energy consumption, while the minimization of maximum speed was also studied. We proposed various lower bounds for the offline and the online settings. In particular, we showed how to use known online algorithms (AVR and BKP) of the classical speed scaling context in the speed scaling with explorable uncertainty setting. Notice also that our approach can directly be applied to the preemptive-non-migratory variant of the problem [GNS14].

5 Uncertain Predictions

Using machine-learned predictions to create algorithms with better approximation guarantees is a very fresh and active field. In this chapter, we study classic scheduling problems under the learning augmented setting. More specifically, we consider the problem of scheduling jobs with arbitrary release dates on a single machine and the problem of scheduling jobs with a common release date on multiple machines. Our objective is to minimize the sum of completion times. For both problems, we propose algorithms which use predictions for taking their decisions. Our algorithms are consistent – i.e. when the predictions are accurate, the performances of our algorithms are close to those of an optimal offline algorithm–, and robust – i.e. when the predictions are wrong, the performance of our algorithms are close to those of an online algorithm without predictions. In addition, we confirm the above theoretical bounds by conducting experimental evaluation comparing the proposed algorithms to the offline optimal ones for both the single and multiple machines settings.

5.1 Formulation of the problem

We are given a set of jobs $\mathcal{J}$ whose actual processing times are not known in advance. However, a predicted value of the processing time is given for each job. Each job is also characterized by a *release time (date).* In the whole extent of this chapter, preemption and migration are allowed at no extra cost. In other words, jobs can be stopped and continue execution in a later time (preemption) and eventually on a different machine (migration). Each machine can execute at most one job at a time. In the first version, we consider scheduling $\mathcal{J}$ on a single machine. Jobs arrive over time, and the algorithm has no prior knowledge on the existence of a job. In the second version, we consider scheduling $\mathcal{J}$ on m identical machines. In this case, we assume that all release times are zero and that the algorithm has knowledge of the total number of jobs as well as their predicted processing times in the beginning. The objective in both problems is to minimize the sum of completion times of the jobs. Our goal is to design algorithms that are both consistent and robust.

Our contribution

A common strategy to obtain these two properties is to construct an algorithm that runs simultaneously a consistent and a robust algorithm, getting in this way the best of

the two worlds. We call such an algorithm *preferential*. In this direction, we provide in Section 5.3 a consistent optimal algorithm, called *Shortest Remaining Predicted Processing Time*, when the predictions are accurate for the single machine problem. By combining this with the non-clairvoyant Round Robin algorithm, we give a preferential algorithm for the problem $1 \mid r_j, \text{non-clair.} \mid \sum_j C_j$. In Section 5.4 we give again a consistent optimal algorithm, called *Shortest Predicted Processing Time First*, when the predictions are accurate for the multiple machines problem. It is also known that Round Robin is 2-competitive for the non-clairvoyant problem in multiple machines [MPT94]. We then give the first preferential algorithm for the problem $P \mid \text{non-clair.} \mid \sum_j C_j$.

5.2 Notations and Preliminaries

We consider a set of n jobs $\mathcal{J}$ to be scheduled either on a single machine or on m parallel machines. Each job $j \in \mathcal{J}$ is characterized by a *release time* r_j and an *actual* (real) processing time p_j. We assume that $p_j > 1 \; \forall j$. For each job, we have also a *predicted* value of its processing time, denoted by y_j. At the release time of a job, the algorithm is informed of the existence of this job as well as of the predicted value of its processing time. The algorithm finds out the actual processing time of the job, only after assigning p_j units of time to the job and hence knows that the job has been completed. In other words, our model is *non-clairvoyant*. A job is considered *active* if it has been released and is not yet completed. Finally, we denote by η_j the error of the prediction for job j, i.e. $\eta_j = |y_j - p_j|$, and $\eta = \sum_j \eta_j$ is the total error of the input.

It is known that the optimal strategy for the objective of minimizing the sum of completion times in the clairvoyant case is to follow the *Shortest Processing Time First (SPT)* rule when all release times are zero. When arbitrary release times are introduced in the model, the *Shortest Remaining Processing Time First (SRPT)* rule is optimal. In the SRPT algorithm, at each time t, the active job with the shortest remaining processing time is chosen to be executed. A job is removed from the active jobs when it has received p_j units of processing time.

Our goal in this chapter is to find algorithms that use the predictions and perform as good as the SPT and SRPT algorithms, in other words optimally, when the predictions are accurate and in the same time not too bad when the predictions are wrong.

A Preferential Algorithm In order to get the best out of each world, we will consider two algorithms. Let $\mathcal{A}$ be a consistent algorithm with a competitive ratio α, and $\mathcal{B}$ be a robust algorithm with a competitive ratio β. Purohit et al. in [PSK18], call a non-clairvoyant scheduling algorithm *monotonic* if it has the following property.

▶ **Definition 5.1 (Monotonicity).** Given two instances with identical inputs and actual job processing times $(p_1, \ldots, p_n)$ and $(p'_1, \ldots, p'_n)$ such that $p_j \leqslant p'_j$ for all j, the

objective function value found by the algorithm for the first instance is no higher than that for the second. ◀

They also give the following lemma on how to combine the two aforementioned monotonic algorithms. For completeness, we provide the proof of the lemma first presented in [PSK18].

▶ **Lemma 5.2.** Given two monotonic algorithms, $\mathcal{A}$ and $\mathcal{B}$, with competitive ratios α and β respectively for the minimum total completion time problem with preemption, and a parameter $\lambda \in (0, 1)$, one can obtain an algorithm with competitive ratio $\min\left\{\frac{\alpha}{\lambda}, \frac{\beta}{1-\lambda}\right\}$. ◀

Proof. The combined algorithm runs the two given algorithms in parallel. The α-approximation is run at a rate of λ, and the β-approximation at a rate of $1-\lambda$. Compared to running at rate 1, if algorithm $\mathcal{A}$ runs at slower rate of λ, all completion times increase by a factor of $\frac{1}{\lambda}$, so it becomes a $\frac{\alpha}{\lambda}$-approximation. Now, the fact that some of the jobs are concurrently being executed by algorithm $\mathcal{B}$ only decreases their processing times from the point of view of $\mathcal{A}$, so by monotonicity, this does not make the objective of $\mathcal{A}$ any worse. Similarly, when algorithm $\mathcal{B}$ runs at a lower rate of $1-\lambda$, it becomes a $\frac{\beta}{1-\lambda}$-approximation, and by monotonicity can only get better from concurrency with $\mathcal{A}$. Thus, both bounds hold simultaneously, and the overall guarantee is their minimum. ■

In what follows, we focus on designing a consistent algorithm when the predictions are good and a robust algorithm otherwise. We will use Lemma 5.2 to get a preferential algorithm.

5.3 Single Machine

In this section, we deal with the scheduling problem with release dates on a single machine. We present the *Shortest Remaining Predicted Processing Time First (SRPPT)* algorithm, and show that it is a consistent algorithm for the objective of minimizing the sum of completion times. We then use the fact that the robust Round Robin algorithm is shown to be 4-competitive for the same objective in a very recent work by Moseley and Vardi [MV22]. Finally, we combine the two results in Theorem 5.8. At the end of the section, we give a simpler analysis for the objective showing that the Round Robin algorithm is 4-approximate in the setting with release dates. We note that our analysis is based on the dual fitting technique as proposed in [GGK+19], using speed augmentation, in contrast with the more tedious analysis based on potential functions in [MV22].

5.3.1 A Consistent Algorithm

We distinguish jobs as overestimated, O, and underestimated ones, $\mathcal{U}$, according to their predicted and actual processing times. More formally, a job is said to be *overestimated* if $y_j \geqslant p_j$. On the other hand, a job is *underestimated* if $y_j < p_j$. Finally, we denote the total error of each subset as $\eta^O = \sum_{j \in O} \eta_j$ for the overestimated jobs, and $\eta^{\mathcal{U}} = \sum_{j \in \mathcal{U}} \eta_j$ for the underestimated ones.

Consider a schedule σ. Given a time t, we denote by $e_j^{[0,t)}$ the *elapsed* time of job j, i.e. the amount of processing time units executed from the beginning of the schedule. We remind that a job j is active at time t if it has been released (i.e. $t \geqslant r_j$), and if its elapsed time is smaller than its actual processing time, $e_j^{[0,t)} < p_j$. We denote by $p_j(t) = p_j - e_j^{[0,t)}$ the *remaining processing time* of an active job j at time t, and by $y_j(t) = y_j - e_j^{[0,t)}$ its *remaining predicted processing time*. To distinguish the completion times of a job j in various schedules we write $C_j\big|_t^\sigma$, where σ denotes the schedule and t the time. Note here that the time subscript is not the same as the completion time of job j. It will be used to distinguish between multiple schedules that change over time. When not necessary and obvious from the context, the time subscript is omitted. In what follows, we may have multiple schedules for a specific time t. In such case, the schedules are created using different instances and hence the distinction between these schedules is clear from the context.

Algorithm

In the *Shortest Remaining Predicted Processing Time First (SRPPT)* algorithm, at each time t, the active job (or one of the active jobs) with the shortest remaining predicted processing time is chosen to be executed. A job is added to the active jobs at time r_j and it is removed from the active jobs after receiving p_j units of processing time. In other words, a job can be removed from the active jobs before receiving y_j units of processing time if it is overestimated. Similarly, if a job is underestimated, it remains active even if it is has received y_j units of time. In this case, at the moment an underestimated job receives y_j units of processing time, it will be executed until completion, i.e. until it has received p_j units of processing time, as the job with the highest priority, having $y_j(t) \leqslant 0$. Only one such job can exist at each time.

To analyze this algorithm, we propose a transformation from an optimal schedule for the sum of completion times to the schedule produced by the SRPPT algorithm. An optimal schedule can be obtain by running the SRPT algorithm using the actual processing times of the jobs. The transformation consists of two steps. The first step (Transformation Step I below) takes place only once in the beginning and its purpose is to bound the cost inflicted by overestimated job in the objective function. The second step (Transformation Step II below), happens iteratively over time starting at time 0. In

this second step, we bound the cost of the underestimated jobs and in the same time we slightly refine the cost of the overestimated jobs in the objective function.

Transformation Step I To bound how much overestimated jobs can increase the value of the objective function, we create an intermediate schedule, called σ^O. Consider the auxiliary instance, I^{aux}, created as follows from an instance I. For the overestimated jobs of I, we include the predicted processing time in the auxiliary instance I^{aux} while for the underestimated jobs of I we include the actual processing time in I^{aux}, i.e. $I^{aux} = \{y_j : j \in O\} \cup \{p_j : j \in \mathcal{U}\}$. Let σ^* be an optimal schedule of this instance, and let j be a job j in this schedule. This job may be preempted several times in the schedule. Denote by $\ell_j^1, \ell_j^2, \ldots, \ell_j^k$ the length of the partial execution of job j in σ^* (with $k \geqslant 1$ and $\sum_k \ell_j^k = p_j$). Underestimated jobs have the same length in the auxiliary instance as in the optimal schedule while overestimated jobs have bigger length in the auxiliary instance. Therefore, for the underestimated jobs, we keep the same length for each part. On the contrary, for the overestimated jobs, we change the length of only the last part and we increase it by its error, i.e. $\ell'^1_j = \ell_j^1, \ell'^2_j = \ell_j^2, \ldots, \ell'^k_j = \ell_j^k + \eta_j$, such as $\sum_k \ell'^k_j = y_j$. As a result the total length of the overestimated job is equal to the predicted value of I^{aux}. We now create the intermediate schedule σ^O, by scheduling the jobs in the auxiliary instance in the same order as in the optimal schedule σ^*, using the partial lengths we created above.

The following lemma gives us a relation between the intermediate and the optimal schedule.

▶ **Lemma 5.3.** We have $\sum_j C_j\big|^{\sigma^O} \leqslant \sum_j C_j\big|^{\sigma^*} + n\eta^O$. ◀

Proof. The processing time of each overestimated job j is increased by η_j and can delay at most n jobs. For a job j, we have:

$$C_j\big|^{\sigma^O} \leqslant C_j\big|^{\sigma^*} + n\eta_j$$

Summing over all jobs, the lemma follows. ■

Example of Step I Consider the following instance. A job is described by a triplet $j = (r_j, p_j, y_j)$. $\mathcal{I} = \{1 = (0, 1, 3), 2 = (0, 2, 2), 3 = (3, 5, 3), 4 = (3, 4, 4), 5 = (4, 1, 1.5), 6 = (7, 2, 2), 7 = (7, 3, 1)\}$. The overestimated jobs are $O = \{1, 2, 4, 5, 6\}$ while the underestimated jobs are $\mathcal{U} = \{3, 7\}$. In Figure 5.1, σ^* is the optimal schedule produced by the SRPT algorithm using the values p_j. Finally, σ^O is the intermediate schedule created as described in Step I of the transformation.

Transformation Step II Let σ^A be the schedule produced by the SRPPT algorithm on instance I. We create an initial auxiliary schedule, σ^{aux}, by applying the SRPT rule

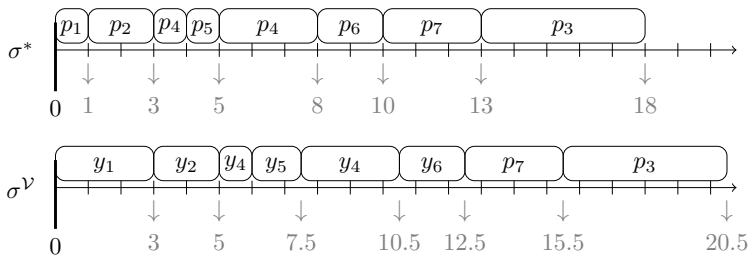


Figure 5.1: An example of the first step of the transformation.

to the intermediate instance, I^{aux}, created in Step I. Then, in what follows, we will transform this initial auxiliary schedule into the one produced by the SRPPT algorithm (σ^A), in a structured way that allow us to calculate the effect of each miss-predicted job in the objective function. We are presenting an example of this transformation in Figure 5.2 and we explain it in the corresponding paragraph.

Before doing this, let us compare the sum of completion times in σ^{aux} and in σ^O. We denote by $C_j\big|_{init}^{\sigma^{aux}}$ the completion time of job j in the initial schedule σ^{aux}. Since both schedules (σ^O and σ^{aux}) use the same instance, I^{aux}, and since σ^{aux} uses the optimal SRPT algorithm, we have:

$$\sum_j C_j\big|_{init}^{\sigma^{aux}} \leqslant \sum_j C_j\big|^{\sigma^O} \tag{5.1}$$

Suppose an auxiliary schedule coincides with σ^A until time t. We denote such a schedule as σ_t^{aux}. We create subsets of the jobs as follows. Denote by $\mathcal{F}^O$ and $\mathcal{F}^{\mathcal{U}}$ the set of overestimated and underestimated jobs, respectively, that finish execution in $[0, t)$. We also create a subset, denoted by $\mathcal{M}^{\mathcal{U}}$, containing the underestimated jobs that were miss-placed in $[0, t)$ (case 3 below). We define the reduced instance at time t as $I_t^{aux} = \{y_j(t) : j \in \mathcal{O}\} \cup \{0 : j \in \mathcal{F}^O\} \cup \{p_j(t) : j \in \mathcal{U}\} \cup \{0 : j \in \mathcal{F}^{\mathcal{U}}\} \cup \{y_j(t) : j \in \mathcal{M}^{\mathcal{U}}\}$. The auxiliary schedule σ_t^{aux} consists of a fixed part which coincides with σ^A in the interval $[0, t)$ and a part which is produced by executing the SRPT algorithm using the reduced instance I_t^{aux}.

Our transformation, starting at time $t = 0$, checks whether the schedules σ^A and σ^{aux} coincide (i.e. schedule the same tasks) until time $t' > t$. If so, we augment the time until the time moment where the two schedules have the first difference. This difference may occur for one of the three following reasons.

1. An overestimated job is completed in σ^A

2. An underestimated job has is completed in σ^{aux}
3. An underestimated job is misplaced in σ^{aux}

The transformation handles each case separately. Before that, we present Lemma 5.4 which has a two-fold meaning. On one hand, it dictates when the second case arises while in the same time, it rules out any other possibility for the two schedules to differ.

▶ **Lemma 5.4.** Let σ_t^{aux} be a schedule that coincides with σ^A until time t, where t is maximal. Let i be the job that starts or continues at time t in σ^A and j be the job that starts or continues at time t in σ_t^{aux}. If $i \neq j$, then i is underestimated. ◀

Proof. Suppose that $i \in O$. For each overestimated job $j \in I_t^{aux}$ its remaining processing time in σ_t^{aux} is equal to its predicted remaining processing time in σ^A. Hence, the job i has the smallest predicted remaining processing time among all jobs from O that can be serviced at the time t. Next, since job i is the job that starts or continues at time t in σ^A then the remaining processing time of i is less than the remaining predicted processing time of each available job from $\mathcal{U}$. Since the actual processing time of any job from $\mathcal{U}$ is greater than its predicted processing time, then job i must precede all available jobs in the schedule σ_t^{aux} at time t. We get a contradiction. Thus, if $i \neq j$ then $i \in \mathcal{U}$. ■

Given that σ^{aux} and σ^A coincide in the interval $[0, t)$, the transformation handles the aforementioned reasons for a difference to exist accordingly.

- **Case 1. [An overestimated job is completed in σ^A]** For a job $j \in O$, suppose l is the largest index of a part of a job that is executed in $[0, t)$. If $\sum_{i=1}^{l} \ell_j^i = p_j$, then remove all parts with indexes $l < i \leqslant k$ from the schedule. Remove job j from the overestimated set, O, and add it to the finished overestimated set, i.e. $\mathcal{F}^O$. Create a new auxiliary schedule by fixing the schedule at time interval $[0, t)$ and complete the rest of the schedule using the SRPT rule with the reduced instance. In this case the objective function decreases by at least η_j. Moreover, each overestimated job completed by time t with $\eta_j > 0$ will create the transformation described in Case 1.
- **Case 2. [An underestimated job has is completed in σ^{aux}]** For a job $j \in \mathcal{U}$. Suppose l is the largest index of a part of a job that is executed in $[0, t)$. If $\sum_{i=1}^{l} \ell_j^i = y_j$ and $\sum_k \ell_j^k = y_j$, then this is the last part of the job to be executed in $[0, t)$ and has finished execution using its predicted time. Note that if this is the case, then job j belongs to set $\mathcal{M}^{\mathcal{U}}$ according to the definition of I_t^{aux}. We can continue the execution of this job until time $t' = t + \eta_j$. Remove job j from the miss-placed underestimated set, $\mathcal{M}^{\mathcal{U}}$, and add it to the finished underestimated set , i.e. $\mathcal{F}^{\mathcal{U}}$. Create a new auxiliary schedule by fixing the schedule at time interval $[0, t')$ and complete the rest of the schedule using the SRPT rule with the reduced instance. In this case the objective function increases by at most $n\eta_j$. Notice, a job that triggers this case, has already been called by case 3, resulting in a total difference of the $(n-1)\eta_j$ in the objective function.

- **Case 3. [An underestimated job is misplaced in σ^{aux}]** The underestimated job j is scheduled in σ^A but not in σ^{aux}. This case arises when another job has shorter remaining processing time than job j. On the other hand, SRPPT chose to execute job j, because its remaining predicted processing time is shorter than any other job. Remove job j from the underestimated set, $\mathcal{U}$, and add it to the miss-placed underestimated set, $\mathcal{M}^{\mathcal{U}}$. This way, we enforce the SRPT algorithm to execute job j first. Create a new auxiliary schedule by fixing the schedule at time interval $[0, t)$ and complete the rest of the schedule using the SRPT rule with the new reduced instance. In this case the objective function decreases by at least η_j. Moreover, if an underestimated job creates this transformation, then in latter time, it creates the transformation described in case 2.

Example of Step II An example of Step II of the transformation is illustrated in Figure 5.2. Starting from the initial auxiliary schedule, we notice that it coincides with σ^A until time instant 3, when an overestimated job finishes execution in σ^{aux}_{init} (Case 1). We remove the part of the job corresponding to the error and create a new auxiliary schedule, σ^{aux}_3, using the remaining instance and the SRPT rule, resulting in the third schedule of the figure. We then observe that an underestimated job, job 3, is misplaced in σ^{aux}_3 (Case 3). In order to force the SRPT rule to place it correctly, we add job 3 in the $\mathcal{M}^{\mathcal{U}}$, and re-run the SRPT algorithm resulting in the fourth schedule in the figure. We next augment the time until time instant 5, where the schedules σ^{aux}_5 and σ^A stop coinciding due to Case 1 again. By removing the error part, η_5, re-running SRPT with the reduced instance and augmenting the time, we come up with schedule σ^{aux}_7. Here, we come across Case 2 where an underestimated job is finished in σ^{aux}_7. We then run job 3 until completion, i.e. until it receives p_3 units of processing time and we augment the time resulting in σ^{aux}_9. Notice here, another underestimated job is misplaced in σ^{aux}_9 (Case 2). We add job 7 in $\mathcal{M}^{\mathcal{U}}$ and re-run the SRPT algorithm. Augmenting the time of the common schedule to time instant 10, we observe that job 7 is finished without interruption. We arrive again at Case 2 where we fix the schedules to coincide until time 12. The final two jobs are correctly placed and exactly predicted, resulting in the last schedule of the figure, σ^{aux}_{final}.

▶ **Lemma 5.5.** We have $\sum_j C_j\big|^{\sigma^A} \leqslant \sum_j C_j\big|^{\sigma^{aux}} - \eta^O + (n-1)\eta^{\mathcal{U}}$. ◀

Proof. Summing up the changes in the objective function for all transformations and taking into account the comments on Cases 1-3, we get

$$\begin{aligned}\sum_j C_j\Big|^{\sigma^A} &\leqslant \sum_j C_j\Big|^{\sigma^{aux}} - \sum_{j\in O}\eta_j + (n-1)\sum_{j\in\mathcal{U}}\eta_j \\ &= \sum_j C_j\Big|^{\sigma^{aux}} - \eta^O + (n-1)\eta^{\mathcal{U}}\end{aligned}$$

■

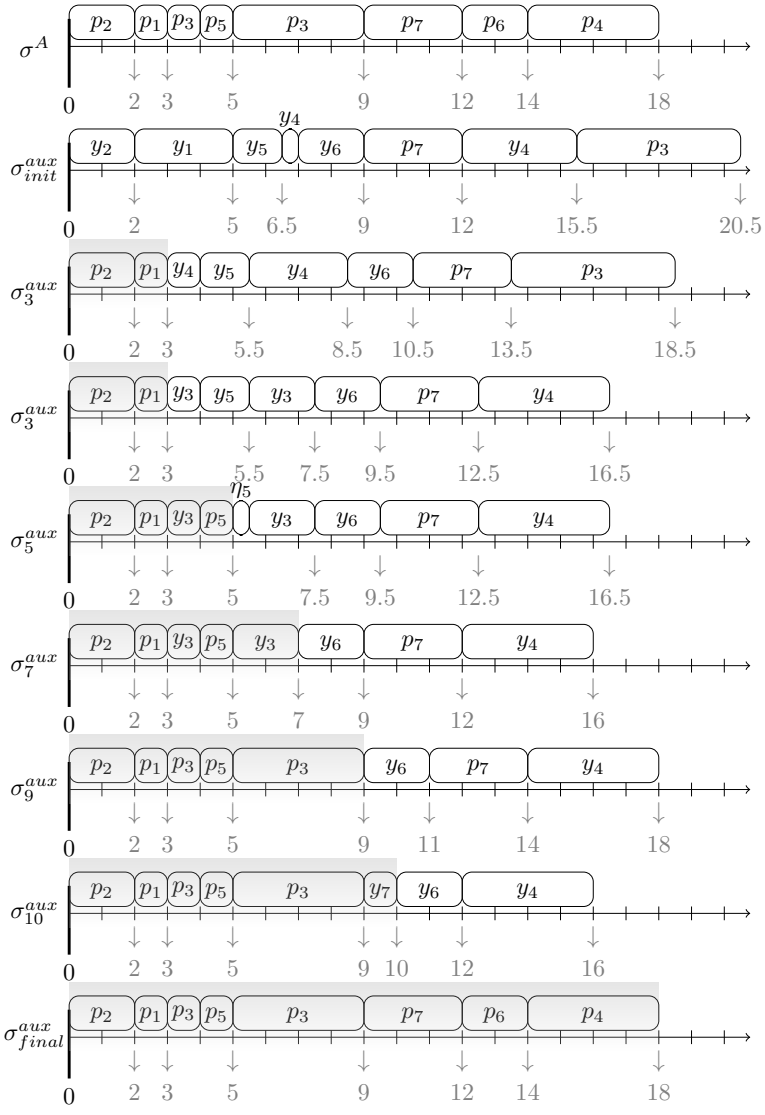


Figure 5.2: An example of the second step of the transformation.

▶ **Theorem 5.6.** The SRPPT algorithm has competitive ratio at most $(1 + \frac{2\eta}{n})$, where $\eta = \sum_j \eta_j$. ◀

Proof. The SRPPT algorithm produces the same schedule as the last iteration of the above described transformation. Using Lemma 5.5 we can bound the performance of the transformation. We get:

$$\begin{aligned} Alg = \sum_j C_j\big|^{\sigma^A} &\leqslant \sum_j C_j\big|^{\sigma^{aux}} - \eta^O + (n-1)\eta^U \\ &\leqslant \sum_j C_j\big|^{\sigma^O} - \eta^O + (n-1)\eta^U \\ &\leqslant \sum_j C_j\big|^{\sigma^*} + n\eta^O - \eta^O + (n-1)\eta^U \\ &\leqslant OPT + (n-1)\eta \end{aligned}$$

Here, the first inequality is given by Lemma 5.5, the second one by Inequality 5.1 and the third by Lemma 5.3.

Given that all processing values are greater or equal than 1, we have that

$$OPT \geqslant \frac{n(n+1)}{2} \tag{5.2}$$

In total, we get

$$\frac{Alg}{OPT} \leqslant 1 + \frac{2(n-1)\eta}{n(n+1)} \leqslant 1 + \frac{2\eta}{n} \tag{5.3}$$

as long as $\frac{n-1}{n+1} < 1$. ■

▶ **Lemma 5.7.** Algorithm SRPPT is monotonic. ◀

Proof. To see that, consider two instances, I and I', with $(p_1, \ldots, p_n)$ and $(p'_1, \ldots, p'_n)$ respectively and $p_j \leqslant p'_j \; \forall \; j$. Both instances have the same predicted values for each job. In other words, the schedules produced by SRPPT are identical. If a job of instance I has a smaller real processing value than the corresponding job of instance I', it will finish earlier, resulting in a smaller completion time which can only reduce the total objective. ■

5.3.2 A Preferential Algorithm

▶ **Theorem 5.8.** The *Preferential Round Robin* algorithm with release dates and parameter $\lambda \in (0,1)$ has competitive ratio at most $\min\left\{\frac{1}{\lambda}(1+\frac{2\eta}{n}), \frac{4}{1-\lambda}\right\}$. In particular, it is $\frac{4}{1-\lambda}$-robust and $\frac{1}{\lambda}$-consistent. ◀

Proof. This follows from the competitive ratio of SRPPT (Lemma 5.6) and the competitive ratio of 4 of Round Robin [MV22]. We combine these two monotonic algorithms using Lemma 5.2. ■

Round Robin – A Simple Analysis

In this subsection, we give a simple analysis showing that the robust Round Robin algorithm is 4-competitive. We remind that this result is included in the more general work of Moseley and Vardi [MV22]. Here, we use a simplified version of the linear program given in [GGK+19] without the precedence constraints. Our analysis is based on the dual fitting technique.

Let $x_j(t)$ be a binary variable which is equal to 1 if the job j is executed at time t and 0 otherwise. We consider the following integer linear program.

$$\min \quad \sum_j \int_0^\infty x_j(t) \cdot \frac{\left(t+\frac{1}{2}\right)}{p_j} dt$$

$$s.t. \quad \int_{r_j}^\infty \frac{x_j(t)}{p_j} dt \geqslant 1 \quad \forall j \tag{5.4}$$

$$\sum_j x_j(t) \leqslant 1 \quad \forall t \geqslant 0 \tag{5.5}$$

$$x_j(t) \in \{0,1\} \quad \forall j, t \geqslant 0 \tag{5.6}$$

The objective of this program corresponds to the fractional completion time criterion, which is a lower bound to our objective. Hence this program is a relaxation for our problem. Constraint (5.4) implies that each job j is processed at least p_j time. Constraint (5.5) implies that at most one job is executed at each time t. By relaxing the integrity Constraint (5.6) for $x_{j,t}$ we get the following dual program.

$$\max \quad \sum_j \alpha_j - \int_0^\infty \beta(t) dt$$

$$s.t. \quad \alpha_j - \beta(t) \cdot p_j \leqslant t + \frac{1}{2} \quad \forall j, t \geqslant r_j \tag{5.7}$$

$$\alpha_j \geqslant 0 \quad \forall j \tag{5.8}$$

$$\beta(t) \geqslant 0 \quad \forall t \geqslant 0 \tag{5.9}$$

In this section we analyze the Round Robin algorithm based on the dual fitting approach using the above primal and dual programs. Our analysis uses speed-augmentation, that is we suppose that in Round Robin the jobs are executed using a bigger speed than in the optimal solution. Specifically, we suppose that the algorithm uses speed 1, while the optimal a speed $\frac{1}{1+\epsilon}$, for any $\epsilon > 0$. In order to simulate this in the primal program, we can focus on Constraint (5.5) and modify it as flowing:

$$\sum_j x_j(t) \leqslant \frac{1}{1+\epsilon} \quad \forall t \geqslant 0$$

implying that the optimal solution cannot execute a whole unit of job at each time t but just a fraction of $\frac{1}{1+\epsilon}$. This modification will also affect the dual objective which becomes:

$$\sum_j \alpha_j - \frac{1}{1+\epsilon} \int_0^\infty \beta(t)dt \tag{5.10}$$

and corresponds to the lower bound to the optimal solution that we will use for the design of our approximation algorithm.

Let now $\overline{C}_j$ be the completion time of the job j in the schedule returned by the Round Robin algorithm. Moreover, let $U(t)$ be the set of unfinished jobs at time t in Round Robin. Note that $U(t)$ contains also the jobs that are not yet released at time t. We assign the dual variables as follows:

- $\alpha_j = \overline{C}_j$, for each j
- $\beta(t) = |U(t)|$, for each $t \geqslant 0$

▶ **Lemma 5.9.** For any job j and any time $t \geqslant r_j$ the dual constraint is satisfied, i.e., $\alpha_j - \beta(t) \cdot p_j \leqslant t + \frac{1}{2}$. ◀

Proof. Fix a job j and a time $t \geqslant r_j$. We denote by $q_\ell(t)$ the remaining processing time of the job ℓ at time t. The Round Robin algorithm will share the time equally to all unfinished jobs. In the worst case for the completion time of j, all unfinished jobs are already released at time t, and hence we have that

$$\begin{aligned}
\overline{C}_j - t &\leqslant \sum_{\ell \in U(t): q_\ell(t) \leqslant q_j(t)} q_\ell(t) + \sum_{\ell \in U(t): q_\ell(t) > q_j(t)} q_j(t) \\
&\leqslant \sum_{\ell \in U(t): q_\ell(t) \leqslant q_j(t)} q_j(t) + \sum_{\ell \in U(t): q_\ell(t) > q_j(t)} q_j(t) \\
&= \sum_{\ell \in U(t)} q_j(t) \leqslant \sum_{\ell \in U(t)} p_j = |U(t)| \cdot p_j
\end{aligned}$$

Therefore,

$$\alpha_j = \overline{C}_j \leqslant t + |U(t)| \cdot p_j = t + \beta(t) \cdot p_j \leqslant t + \frac{1}{2} + \beta(t) \cdot p_j$$

and the lemma follows. ■

▶ **Theorem 5.10.** The Round Robin algorithm is $(1+\epsilon)$-speed $\frac{1+\epsilon}{\epsilon}$-competitive with respect to the $\sum C_j$ objective. ◀

Proof. By using the proposed assignment of dual variable, for the dual objective using speed augmentation (Expression (5.10)) we get:

$$\sum_j \alpha_j - \frac{1}{1+\epsilon} \int_0^\infty \beta(t)dt = \sum_j \overline{C}_j - \frac{1}{1+\epsilon} \int_0^\infty |U(t)|dt$$

Note that, $\sum_j \overline{C}_j$ is the objective value of the Round Robin algorithm. Since $U(t)$ contains also the jobs that are not released at t, the integral in this equation corresponds also to the objective of Round Robin, i.e., $\int_0^\infty |U(t)|dt = \sum_j \overline{C}_j$. Thus for the dual objective is equal to

$$\sum_j \overline{C}_j - \frac{1}{1+\epsilon} \sum_j \overline{C}_j = \frac{\epsilon}{1+\epsilon} \sum_j \overline{C}_j$$

and hence the competitive ratio of Round Robin is $\frac{1+\epsilon}{\epsilon}$. ■

▶ **Theorem 5.11.** The Round Robin algorithm is 4-competitive with respect to the $\sum C_j$ objective. ◀

Proof. Based on the Theorem 5.10 and the well-known generic transformation of an s-speed ρ-competitive algorithm to a $s\rho$-competitive algorithm by [BP04], we have that the Round Robin has an approximation ratio of $\frac{(1+\epsilon)^2}{\epsilon}$. This is minimized by setting the user defined parameter ϵ equal to 1, and hence we get a competitive ratio equal to 4. ■

5.4 Multiple Machines

In this section we consider the problem of scheduling jobs on a set of m identical multiple machines using predictions. Let $\mathcal{J}$ be an instance of n jobs with the execution time of each job being $p_j : 1 \leqslant j \leqslant n$. Each job has also a *predicted* execution time, $y_j : 1 \leqslant j \leqslant n$. Similarly to the previous section, our model is *non-clairvoyant*, meaning that the actual processing time is revealed to the algorithm only when p_j units of processing time are assigned to a specific job. In this section we consider the version of the problem where all jobs have release dates equal to 0. In what follows, jobs can be preempted

and migrated to a different machine at no cost at any time. Our goal is to design an algorithm taking into account the predictions under the objective of minimizing the total sum of completion times.

We present the consistent algorithm *Shortest Predicted Processing Time First for Multiple Machines (SPPT(m))*. We show that SPPT(m) is optimal if the predictions are accurate. For this specific setting, Round Robin is shown to be 2-competitive in [MPT94]. Finally, we combine the two results in Theorem 5.15.

5.4.1 A Consistent Algorithm

We start by defining a grouping between the jobs using their predicted processing times.

▶ **Definition 5.12.** Let $\mathcal{J}$ be any instance of size n with the predicted processing times of the jobs being such that $y_1 \geqslant y_2 \geqslant \ldots \geqslant y_n$. For $1 \leqslant k \leqslant \lceil \frac{n}{m} \rceil$, we define the group of jobs $G_{Pr}(k)$ as $G_{Pr}(k) = \{y_i | (k-1)m < i \leqslant km\}$. ◀

Groups give the execution ordering of the jobs in each machine. A job of group 1 is scheduled on a machine in the last position, a job of group 2 is scheduled second to last, etc. All groups, except possibly the last one, consist of m consecutive jobs.

Optimal To bound the value of an optimal solution, consider a grouping based on the actual processing time of the jobs, arranged so as $p_1 \geqslant p_2 \geqslant \ldots \geqslant p_n$. For $1 \leqslant k \leqslant \lceil \frac{n}{m} \rceil$, define $G(k)$ as $G(k) = \{p_i | (k-1)m < i \leqslant km\}$. This grouping was first introduced by Bruno et al. in [BCS74]. A job in group $G(k)$ has exactly $k-1$ jobs scheduled after it on its machine. Thus, it contributes to the completion time of exactly k jobs.

$$OPT = \sum_{k=1}^{\lceil \frac{n}{m} \rceil} \sum_{i \in G(k)} k p_i \geqslant \frac{\lceil \frac{n}{m} \rceil (\lceil \frac{n}{m} \rceil + 1)}{2} \geqslant \frac{n(n+m)}{2m^2} \tag{5.11}$$

The inequality holds due to the assumption $p_i \geqslant 1$ for each job i.

It is known that the SPT algorithm returns a schedule which minimizes the sum of completion times. We can decompose the impact of a job in this sum in two parts. The first one represents the actual processing time of each job while the second one is the amount this job has been delayed by other jobs executed before in the same machine. Summing the first and second part over all jobs, we get the following:

$$OPT \geqslant \sum_{i=1}^{n} p_i + \frac{1}{m} \sum_{\substack{(i,j): p_i < p_j \\ i \in G_{Pr}(k), j \in G_{Pr}(l), k \neq l}} p_i \tag{5.12}$$

Note the second sum. For any pair of jobs (i, j), we count the length of job i only if its real processing time is smaller than the real processing time of job j. Furthermore, we make sure that two jobs of the pair do not belong to the same group. Finally, note that the subscript of the second sum counts the length of a job for all jobs that belong in a different group, i.e. m different jobs, as many as the jobs in a group. However, a given job actually delays only the jobs scheduled in the same machine. To correctly measure the delay, we have to normalize the second sum by m.

Algorithm

The SPPT(m) algorithm first creates $\lceil n/m \rceil$ groups $G_{Pr}(k)$ of the tasks, considering the tasks sorted in non increasing order of their predicted values (as seen above, $G_{Pr}(1)$ contains the m tasks with the largest predicted values, and so forth). Then, each machine receives (at most) one task of each group. On each machine, the tasks are scheduled in a non decreasing order of their predicted values, and the processing times of each task is its actual processing time.

▶ **Theorem 5.13.** The SPPT(m) algorithm has competitive ratio at most $1 + \frac{2m\eta}{n}$, where $\eta = \sum_j \eta_j$. ◀

Proof. A job i that belongs in group $G_{Pr}(k)$ will delay all the jobs that are scheduled in the same machine as it is. More specifically, it will delay those jobs, j, that belong in a different group $G_{Pr}(l)$ with $l < k$. For any pair of jobs i, j such that $i \in G_{Pr}(k)$ and $j \in G_{Pr}(l)$, we define as $d(i, j)$ the amount of job i that has been executed before the completion time of job j. We can define now the performance of the algorithm as follows:

$$Alg = \sum_{j=1}^{n} p_j + \frac{1}{m} \sum_{\substack{(i,j):p_i<p_j \\ i\in G_{Pr}(k), j\in G_{Pr}(l)}} d(i, j) \tag{5.13}$$

In the second sum, we count the delay for all the jobs that belong in a different group. However, only one of them is actually delayed, i.e. the one scheduled in the same machine. In order to count the delay correctly, we divide the sum by the number of machines. If job i is correctly predicted, i.e. $y_i < y_j$ and $p_i < p_j$, then job i delays job j for x_i amount of time. If job i is wrongly predicted, i.e. $y_i > y_j$ and $p_i < p_j$, then job j delays job i for x_j amount of time. If jobs i and j belong in the same group, they are executed in different machines, therefore no pairwise delay exists. More formally, given the group that each job belongs to, for a pair of jobs (i, j) such that job i belongs to

group $G_{Pr}(k)$ and j belongs to group $G_{Pr}(l)$ and $p_i < p_j$, we define the delay as follows:

$$d(i,j) = \begin{cases} p_i & \text{if } k > l \\ 0 & \text{if } k = l \\ p_j & \text{if } k < l \end{cases} \tag{5.14}$$

We can now split the second sum of the performance of the algorithm according to the correctness of the prediction.

$$\begin{aligned}
Alg &= \sum_{j=1}^{n} p_j + \frac{1}{m} \sum_{\substack{(i,j):p_i<p_j \\ i\in G_{Pr}(k), j\in G_{Pr}(l) \\ k>l}} p_i + \frac{1}{m} \sum_{\substack{(i,j):p_i<p_j \\ i\in G_{Pr}(k), j\in G_{Pr}(l) \\ k<l}} p_i \\
&= \sum_{j=1}^{n} p_j + \frac{1}{m} \sum_{\substack{(i,j):p_i<p_j \\ i\in G_{Pr}(k), j\in G_{Pr}(l) \\ k\neq l}} p_i + \frac{1}{m} \sum_{\substack{(i,j):p_i<p_j \\ i\in G_{Pr}(k), j\in G_{Pr}(l) \\ k<l}} (p_j - p_i) \\
&\leqslant \sum_{j=1}^{n} p_j + \frac{1}{m} \sum_{\substack{(i,j):p_i<p_j \\ i\in G_{Pr}(k), j\in G_{Pr}(l) \\ k\neq l}} p_i + \frac{1}{m} \sum_{\substack{(i,j):p_i<p_j \\ i\in G_{Pr}(k), j\in G_{Pr}(l) \\ k<l}} (p_j - y_j) + (y_i - p_i) \\
&\leqslant \sum_{j=1}^{n} p_j + \frac{1}{m} \sum_{\substack{(i,j):p_i<p_j \\ i\in G_{Pr}(k), j\in G_{Pr}(l) \\ k\neq l}} p_i + \frac{1}{m} \sum_{\substack{(i,j):p_i<p_j \\ i\in G_{Pr}(k), j\in G_{Pr}(l) \\ k<l}} |p_j - y_j| + |y_i - p_i| \\
&= \sum_{j=1}^{n} p_j + \frac{1}{m} \sum_{\substack{(i,j):p_i<p_j \\ i\in G_{Pr}(k), j\in G_{Pr}(l) \\ k\neq l}} p_i + \frac{1}{m} \sum_{\substack{(i,j):p_i<p_j \\ i\in G_{Pr}(k), j\in G_{Pr}(l) \\ k<l}} (\eta_i + \eta_j) \qquad (5.15) \\
&\leqslant OPT + \frac{1}{m} \sum_{\substack{(i,j):p_i<p_j \\ i\in G_{Pr}(k), j\in G_{Pr}(l) \\ k<l}} (\eta_i + \eta_j) \\
&\leqslant OPT + \frac{(n-1)}{m}\eta \qquad (5.16)
\end{aligned}$$

In addition, to pass from the exact values to the error, we use the fact that $y_i > y_j$ in the specific sum and the definition of the error, i.e., $\eta_j = |p_j - y_j|$. In (5.15) we use inequality

(5.12). From (5.16) and inequality (5.11) we get:

$$\frac{Alg}{OPT} \leqslant 1 + \frac{(n-1)\eta}{mOPT} = 1 + \frac{2m(n-1)\eta}{n(n+m)} < 1 + \frac{2m\eta}{n}$$

as long as $\frac{n-1}{n+m} < 1$ due to $m \geqslant 1$ and the proof is complete. ■

▶ **Lemma 5.14.** Algorithm SPPT(m) is monotonic. ◀

Proof. (Same as Lemma 5.7) To see that, consider two instances, I and I', with $(p_1, \ldots, p_n)$ and $(p'_1, \ldots, p'_n)$ respectively and $p_j \leqslant p'_j \; \forall \; j$. Both instances have the same predicted values for each job. In other words, the schedules produced by SPPT(m) are identical. If a job of instance I has a smaller real processing value than the corresponding job of instance I', it will finish earlier, resulting in a smaller completion time which can only reduce the total objective. ■

5.4.2 A Preferential Algorithm

▶ **Theorem 5.15.** The *Preferential Round Robin for multiple machines* algorithm with parameter $\lambda \in (0, 1)$ has competitive ratio at most $\min\left\{\frac{1}{\lambda}(1 + \frac{2m\eta}{n}), \frac{2}{1-\lambda}\right\}$. In particular, it is $\frac{2}{1-\lambda}$-robust and $\frac{1}{\lambda}$-consistent. ◀

Proof. This follows from the competitive ratio of SPPT(m) (Theorem 5.13) and the competitive ratio of 2 of Round Robin [MPT94]. We combine these two monotonic algorithms using Lemma 5.2. ■

5.5 Experimental Evaluation

In this section, we present experimental results which confirm the bounds stated on Theorems 5.8 and 5.15 for the single and multiple machines respectively. The code is publicly available at https://github.com/ildoge/SUP.

We create artificial instances, each one consisting of $n = 50$ jobs for the single machine case, following the same approach as in [PSK18]. We draw the actual processing time values, p_j, independently for each job from a Pareto distribution with a parameter $\alpha = 1.1$. An error value, η_j, is drawn from a normal distribution with mean 0 and standard deviation σ. Finally, we set the predicted processing time of each job to be $y_j = p_j + \eta_j$. For the release times of the jobs, we first set the interval $[0, \tau \sum_j p_j]$ using the actual processing times created before, where $\tau \geqslant 0$ is a parameter that allows us to scale the size of the interval. We then draw n values uniformly at random from this interval and assign them to the n jobs. Notice that, when the release dates are dense (jobs become available close to each other), then there is not a lot of idle time in the

schedule. On the contrary, if jobs become available sparsely, then both the algorithm and the optimal are forced to take similar scheduling decisions. For this reason, we set $\tau = 0.1$ for the first simulations.

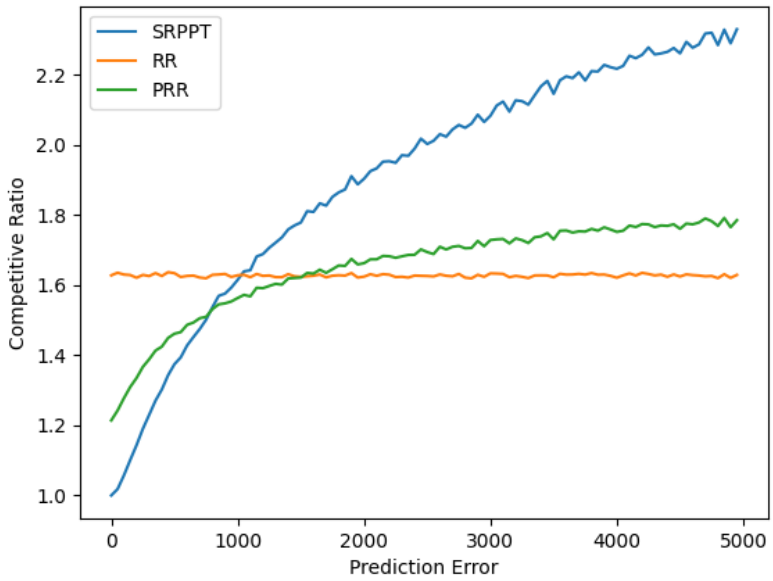


Figure 5.3: Single machine, $\tau = 0.1$

We implement the algorithms: Round Robin (RR), Shortest Remaining Predicted Processing Time First (SRPPT) and Preferential Round Robin(PRR). We compare the performance of the algorithms for different values of error by parameterizing the σ of the normal distribution. We use values of σ that belong in $[0, 5000]$ using a step of 50. The performance of the algorithms is compared to the optimal Shortest Remaining Processing Time First (SRPT) algorithm using the actual processing values. The ratio of an algorithm is taken as the average over 2000 independent runs. As we see in Figure 5.3, the SRPPT algorithm performs really well when the error is small enough, but deteriorates fast otherwise. We observe also the robust behavior of Round Robin independently of the total error. In between, we have our Preferential Round Robin algorithm, with $\lambda = 1/2$, which performs well for small values of error while remaining competitive to Round Robin when the error is big.

Notice that for $\sigma = 1000$, our Preferential Round Robin algorithm out-performs both the SRPPT and RR. By fixing this value of σ, we compare the performance of PRR for

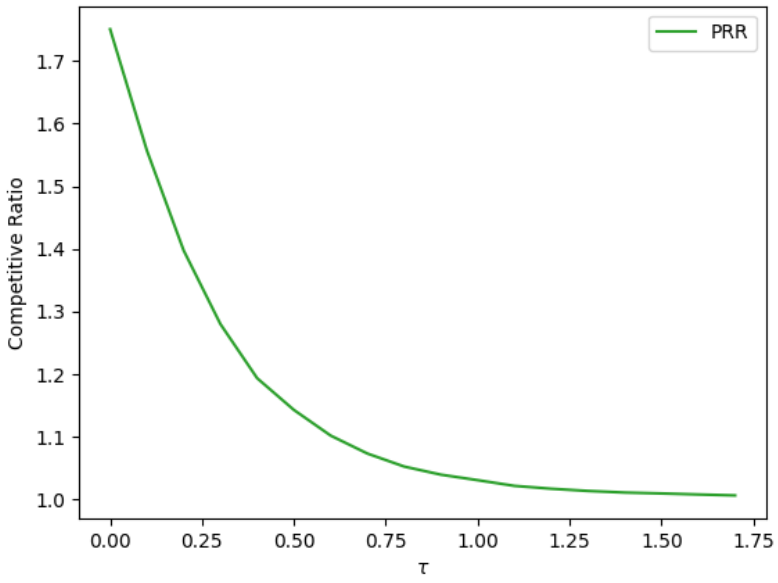


Figure 5.4: Single machine, $\sigma = 1000$

various values of the parameter τ (Figure 5.4). As expected, the ratio of the algorithm deteriorates as the interval becomes smaller. Interestingly, we observe that we get the worst case ratios when all release times are equal to zero. On the other hand, when the interval is large and the release times sparsely distributed in it, our algorithm is nearly optimal.

For the multiple machines setting, we implement the algorithms: Round Robin (RR(m)), Shortest Predicted Processing Time First (SPPT(m)) and Preferential Round Robin (PRR(m)). We follow the same approach to produce the actual processing values, the error and the predicted processing values. In this case, we have no release times. Here, we follow the same framework of executions as before, however we compare each algorithm to the optimal Shortest Processing Time First (SPT(m)) algorithm for multiple machines. We note that we run these experiments using $m = 5$ machines and $n = 250$ jobs in order to maintain enough jobs and load in each machine. The results can be seen in Figure 5.5 and can be interpreted in the same way as the results of the single machine setting.

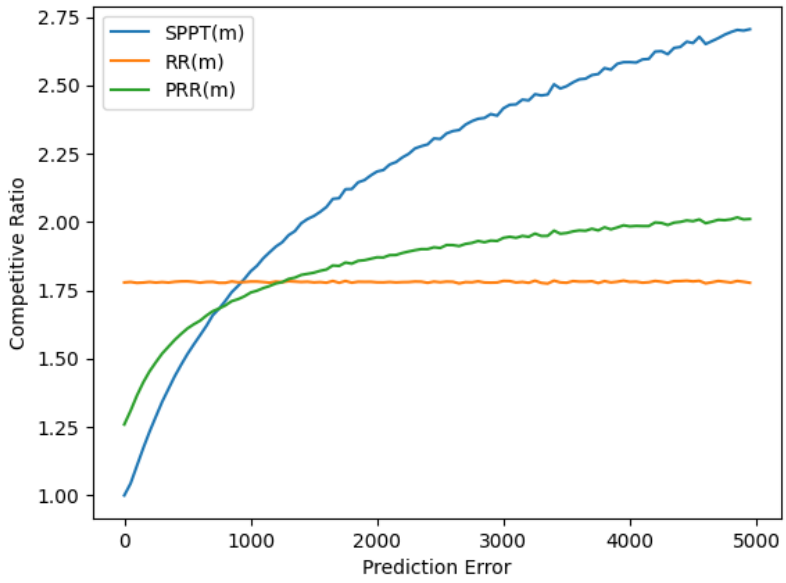


Figure 5.5: Multiple machines

5.6 Conclusion

In this chapter, we studied the problem of integrating predictions to improve the performance of online algorithms for the non-clairvoyant scheduling problem on a single machine with release dates and on multiple machines without release dates. Using predictions to improve the performance of algorithms is a very versatile technique and can be applied to various fields. Today, machine learning algorithms can give predictions on the length of the jobs (or other characteristics) that will arrive, justifying the attention to the novel technique.

6 General Conclusions and Outlook

The need for more energy efficient scheduling algorithms is of paramount importance nowadays. In this thesis we studied a plethora of scheduling models under various constraints that are a step closer to more realistic platforms. The proposed algorithms minimize the energy consumption either explicitly as an objective function, or implicitly by imposing constraints to reduce the energy-hungry data movement.

In our first model, studied in Chapter 3, we considered two kind of machines, i.e. computing and input/output nodes, motivated by real-life high performance computers. Applying contiguity and locality constraints, we proposed approximation algorithms for malleable jobs whose processing time is defined by either a proportional or a generic speed-up function. We assumed that the machines are interconnected in a line topology network. Even though the line grasps the basic characteristics of direct architectures, it is too simplistic compared to current implementations. It would be very interesting to see if our approach can be extended to higher dimension topologies. In addition, one can extend the discussion on indirect topologies and spawn new constraints to reduce data movement. Furthermore, in our model, each job needs a specific input/output node for its execution. The model will acquire a different perspective if jobs dictate a set of input/output nodes that they can use, and the schedule may allocate one or more of them. This still allows to minimize data movement but this may help to optimize better the objective function. In the same vein, another possible direction could be to relax the locality and/or contiguity constraints in order to study the trade-off between our objective function (e.g. makespan) and the "degree of locality". This could lead to a bi-objective problem. As HPC users submit jobs in an online manner, it would be interesting to either design purely online algorithms for the same problem or find an efficient way to transform already known offline algorithms to online ones. Finally, one can implement the proposed algorithms and evaluate them in practice.

Next, in Chapter 4, we studied the model of scheduling jobs on a single (or multiple) speed scalable machine(s) under the explorable uncertainty with tests setting. The scheduler here has to take two critical decisions about executing or not the tests and adjusting the speed of the machine(s), which is directly related to the energy consumption. We started by applying restrictions on the deadlines of the jobs when they all arrive at the same time, and we finished with the general online case with arbitrary deadlines. We proposed competitive algorithms for all cases. Adding tests that can reveal information for the input and having a direct impact on the objective function is

a fairly new field. We are curious whether the new framework can work with other existing online algorithms, such as the Optimal Available (OA) proposed in [YDS95]. Determining the impact of testing (if possible) in the case of the OA is an interesting question. Moreover, in our approach, we consider that the complete workload of the upper bound is executed if the test is not made. An interesting alternative would be to assume that the actual workload of a job is fixed with or without the test. In such case, if the test is not made, the job would be finished after its exact workload is executed (and not its upper bound). Here, the test would be used to reveal the exact workload of a task before its execution, and thus allow the scheduler to adjust the speed accordingly, in order to minimize energy consumption.

Finally, in Chapter 5, we studied the problem of scheduling jobs with unknown processing times using uncertain predictions on a single (or multiple) machine(s). The use of (potentially erroneous) predicted values in the place of processing times allows us to design algorithms that are both consistent and robust. There is a trade-off between the accuracy of the predictions and the efficiency of the online algorithm. However, the upper bound in case of inaccurate predictions, is not far from the best non-clairvoyant algorithm. Our approach, essentially, combines a consistent and a robust algorithm in order to achieve the best of both worlds. The methodology of combining algorithms is not well studied and is an interesting point where our approach can gain in performance. The domain of augmenting problems with the use of predictions is very recent and our work is one of the first steps to apply it in scheduling problems. One could study other variants of scheduling under the new framework such as associating weights to jobs and using parallel jobs (either rigid or malleable). Other objective functions could be also considered.

Bibliography

[AAG15] Susanne Albers, Antonios Antoniadis, and Gero Greiner. **On multi-processor speed scaling with migration**. *J. Comput. Syst. Sci.* 81:7 (2015), 1194–1209. URL: https://doi.org/10.1016/j.jcss.2015.03.001 (see pages 12–14, 62).

[ABC+99] Foto N. Afrati, Evripidis Bampis, Chandra Chekuri, David R. Karger, Claire Kenyon, Sanjeev Khanna, Ioannis Milis, Maurice Queyranne, Martin Skutella, Clifford Stein, and Maxim Sviridenko. **Approximation Schemes for Minimizing Average Weighted Completion Time with Release Dates**. In: *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*. IEEE Computer Society, 1999, 32–44. URL: https://doi.org/10.1109/SFFCS.1999.814574 (see page 16).

[ABK+02] Abdel Krim Amoura, Evripidis Bampis, Claire Kenyon, and Yannis Manoussakis. **Scheduling Independent Multiprocessor Tasks**. *Algorithmica* 32:2 (2002), 247–261. URL: https://doi.org/10.1007/s00453-001-0076-9 (see pages 10, 11).

[ABK+18] Luciana Arantes, Evripidis Bampis, Alexander V. Kononov, Manthos Letsios, Giorgio Lucarelli, and Pierre Sens. **Scheduling under Uncertainty: A Query-based Approach**. In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*. Ed. by Jérôme Lang. ijcai.org, 2018, 4646–4652. URL: https://doi.org/10.24963/ijcai.2018/646 (see page 15).

[ABK+19] Eric Angel, Evripidis Bampis, Fadi Kacem, and Dimitrios Letsios. **Speed scaling on parallel processors with migration**. *J. Comb. Optim.* 37:4 (2019), 1266–1282. URL: https://doi.org/10.1007/s10878-018-0352-0 (see pages 12, 13).

[ABL+17] Susanne Albers, Evripidis Bampis, Dimitrios Letsios, Giorgio Lucarelli, and Richard Stotz. **Scheduling on power-heterogeneous processors**. *Inf. Comput.* 257 (2017), 22–33. URL: https://doi.org/10.1016/j.ic.2017.09.013 (see pages 12–14).

[ACE+20] Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. **Online metric algorithms with untrusted predictions**. In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, 345–355. URL: http://proceedings.mlr.press/v119/antoniadis20a.html (see page 17).

[AE20] Susanne Albers and Alexander Eckl. **Explorable Uncertainty in Scheduling with Non-Uniform Testing Times**. *CoRR* abs/2009.13316 (2020). arXiv: 2009.13316. URL: https://arxiv.org/abs/2009.13316 (see pages 2, 15).

[AGP20] Keerti Anand, Rong Ge, and Debmalya Panigrahi. **Customizing ML Predictions for Online Algorithms**. In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, 303–313. URL: http://proceedings.mlr.press/v119/anand20a.html (see page 17).

[Alb10] Susanne Albers. **Energy-efficient algorithms**. *Commun. ACM* 53:5 (2010), 86–96. URL: https://doi.org/10.1145/1735223.1735245 (see page 12).

[AMS14] Susanne Albers, Fabian Müller, and Swen Schmelzer. **Speed Scaling on Parallel Processors**. *Algorithmica* 68:2 (2014), 404–425. URL: https://doi.org/10.1007/s00453-012-9678-7 (see pages 12, 13).

[Bam16] Evripidis Bampis. **Algorithmic Issues in Energy-Efficient Computation**. In: *Discrete Optimization and Operations Research - 9th International Conference, DOOR 2016, Vladivostok, Russia, September 19-23, 2016, Proceedings*. Ed. by Yury Kochetov, Michael Khachay, Vladimir L. Beresnev, Evgeni A. Nurminski, and Panos M. Pardalos. Vol. 9869. Lecture Notes in Computer Science. Springer, 2016, 3–14. URL: https://doi.org/10.1007/978-3-319-44914-2%5C_1 (see page 12).

[BBC+11] Nikhil Bansal, David P. Bunde, Ho-Leung Chan, and Kirk Pruhs. **Average Rate Speed Scaling**. *Algorithmica* 60:4 (2011), 877–889. URL: https://doi.org/10.1007/s00453-009-9379-z (see pages 12, 13).

[BCS74] J. Bruno, E.G. Coffman, and R. Sethi. **Scheduling Independant Tasks to Reduce Mean Finishing Time**. *Communications of the A.C.M.* 17:7 (1974) (see page 80).

[BDD+92] Jacek Blazewicz, Paolo Dell'Olmo, Maciej Drozdowski, and Maria Grazia Speranza. **Scheduling Multiprocessor Tasks on Three Dedicated Processors**. *Inf. Process. Lett.* 41:5 (1992), 275–280. URL: https://doi.org/10.1016/0020-0190(92)90172-R (see page 10).

[BDG+15] Iwo Bladek, Maciej Drozdowski, Frédéric Guinand, and Xavier Schepler. **On contiguous and non-contiguous parallel task scheduling**. *J. Scheduling* 18:5 (2015), 487–495. URL: https://doi.org/10.1007/s10951-015-0427-z (see pages 10, 12).

[BDJ+09] Marin Bougeret, Pierre-François Dutot, Klaus Jansen, Christina Otte, and Denis Trystram. **Approximation Algorithms for Multiple Strip Packing**. In: *Approximation and Online Algorithms, 7th International Workshop, WAOA 2009, Copenhagen, Denmark, September 10-11, 2009. Revised Papers*. Ed. by Evripidis Bampis and Klaus Jansen. Vol. 5893. Lecture Notes in Computer Science. Springer, 2009, 37–48. URL: https://doi.org/10.1007/978-3-642-12450-1%5C_4 (see page 11).

[BDK+20] Evripidis Bampis, Konstantinos Dogeas, Alexander V. Kononov, Giorgio Lucarelli, and Fanny Pascual. **Scheduling Malleable Jobs Under Topological Constraints**. In: *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), New Orleans, LA, USA, May 18-22, 2020*. IEEE, 2020, 316–325. DOI: 10.1109/IPDPS47924.2020.00041. URL: https://doi.org/10.1109/IPDPS47924.2020.00041 (see page 6).

[BDK+21] Evripidis Bampis, Konstantinos Dogeas, Alexander V. Kononov, Giorgio Lucarelli, and Fanny Pascual. **Speed Scaling with Explorable Uncertainty**. In: *SPAA '21: 33rd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, 6-8 July, 2021*. Ed. by Kunal Agrawal and Yossi Azar. ACM, 2021, 83–93. DOI: 10.1145/3409964.3461812. URL: https://doi.org/10.1145/3409964.3461812 (see page 6).

[BDL+18] Raphaël Bleuse, Konstantinos Dogeas, Giorgio Lucarelli, Grégory Mounié, and Denis Trystram. **Interference-Aware Scheduling Using Geometric Constraints**. In: *Euro-Par 2018: Parallel Processing - 24th International Conference on Parallel and Distributed Computing, Turin, Italy, August 27-31, 2018, Proceedings*. 2018, 205–217. URL: https://doi.org/10.1007/978-3-319-96983-1%5C_15 (see pages 12, 19, 21, 22, 25, 32, 33).

[BDW86] Jacek Blazewicz, Mieczyslaw Drabowski, and Jan Weglarz. **Scheduling Multiprocessor Tasks to Minimize Schedule Length**. *IEEE Trans. Computers* 35:5 (1986), 389–393. URL: https://doi.org/10.1109/TC.1986.1676781 (see page 9).

[BG08] Brad D. Bingham and Mark R. Greenstreet. **Energy Optimal Scheduling on Multiprocessors with Migration**. In: *IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA 2008, Sydney, NSW, Australia, December 10-12, 2008*. IEEE Computer Society, 2008, 153–161. URL: https://doi.org/10.1109/ISPA.2008.128 (see page 13).

[BKK+04] Adam L. Buchsbaum, Howard J. Karloff, Claire Kenyon, Nick Reingold, and Mikkel Thorup. **OPT Versus LOAD in Dynamic Storage Allocation**. *SIAM J. Comput.* 33:3 (2004), 632–646. URL: https://doi.org/10.1137/S0097539703423941 (see page 11).

[BKL+15] Evripidis Bampis, Alexander V. Kononov, Dimitrios Letsios, Giorgio Lucarelli, and Ioannis Nemparis. **From preemptive to non-preemptive speed-scaling scheduling**. *Discret. Appl. Math.* 181 (2015), 11–20. URL: https://doi.org/10.1016/j.dam.2014.10.007 (see page 12).

[BKL+18] Evripidis Bampis, Alexander V. Kononov, Dimitrios Letsios, Giorgio Lucarelli, and Maxim Sviridenko. **Energy-efficient scheduling and routing via randomized rounding**. *J. Sched.* 21:1 (2018), 35–51. URL: https://doi.org/10.1007/s10951-016-0500-2 (see page 12).

[BKP07] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. **Speed scaling to manage energy and temperature**. *J. ACM* 54:1 (2007), 3:1–3:39. URL: https://doi.org/10.1145/1206035.1206038 (see pages 12, 13, 57–59).

[BLL15] Evripidis Bampis, Dimitrios Letsios, and Giorgio Lucarelli. **Green scheduling, flows and matchings**. *Theor. Comput. Sci.* 579 (2015), 126–136. URL: https://doi.org/10.1016/j.tcs.2015.02.020 (see page 12).

[BLT18] Raphaël Bleuse, Giorgio Lucarelli, and Denis Trystram. **A Methodology for Handling Data Movements by Anticipation: Position Paper**. In: *Euro-Par 2018: Parallel Processing Workshops - Euro-Par 2018 International Workshops, Turin, Italy, August 27-28, 2018, Revised Selected Papers*. 2018, 134–145. URL: https://doi.org/10.1007/978-3-030-10549-5%5C_11 (see pages 12, 21).

[BMR+20] Étienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. **Learning Augmented Energy Minimization via Speed Scaling**. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/af94ed0d6f5acc95f97170e3685f16c0-Abstract.html (see page 17).

[BMS20] Étienne Bamas, Andreas Maggiori, and Ola Svensson. **The Primal-Dual method for Learning Augmented Algorithms**. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. 2020. URL: https://proceedings.neurips.cc/

paper/2020/hash/e834cb114d33f729dbc9c7fb0c6bb607-Abstract.html (see pages 6, 17, 18).

[BP04] Nikhil Bansal and Kirk Pruhs. **Server Scheduling in the Weighted l_p Norm**. In: *LATIN 2004: Theoretical Informatics, 6th Latin American Symposium, Buenos Aires, Argentina, April 5-8, 2004, Proceedings*. Ed. by Martin Farach-Colton. Vol. 2976. Lecture Notes in Computer Science. Springer, 2004, 434–443. URL: https://doi.org/10.1007/978-3-540-24698-5%5C_47 (see page 79).

[BR70] George Bozoki and Jean-Paul Richard. **A Branch-and-Bound Algorithm for the Continuous-Process Job-Shop Scheduling Problem**. *A I I E Transactions* 2:3 (1970), 246–252. URL: https://doi.org/10.1080/05695557008974759 (see page 10).

[CM01] Jianer Chen and Antonio Miranda. **A Polynomial Time Approximation Scheme for General Multiprocessor Job Scheduling**. *SIAM J. Comput.* 31:1 (2001), 1–17. URL: https://doi.org/10.1137/S0097539798348110 (see page 10).

[CMN+01] Chandra Chekuri, Rajeev Motwani, B. Natarajan, and Clifford Stein. **Approximation Techniques for Average Completion Time Scheduling**. *SIAM J. Comput.* 31:1 (2001), 146–166. URL: https://doi.org/10.1137/S0097539797327180 (see page 16).

[DEM+18] Christoph Dürr, Thomas Erlebach, Nicole Megow, and Julie Meißner. **Scheduling with Explorable Uncertainty**. In: *ITCS 2018*. 2018, 30:1–30:14. URL: https://doi.org/10.4230/LIPIcs.ITCS.2018.30 (see pages 2, 3, 15).

[DIR+20] Yihe Dong, Piotr Indyk, Ilya P. Razenshteyn, and Tal Wagner. **Learning Space Partitions for Nearest Neighbor Search**. In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=rkenmREFDr (see page 17).

[DL89] Jianzhong Du and Joseph Y.-T. Leung. **Complexity of Scheduling Parallel Task Systems**. *SIAM J. Discrete Math.* 2:4 (1989), 473–487. URL: https://doi.org/10.1137/0402042 (see pages 9, 10).

[DL93] Jianzhong Du and Joseph Y.-T. Leung. **Minimizing Mean Flow Time with Release Time and Deadline Constraints**. *J. Algorithms* 14:1 (1993), 45–68. URL: https://doi.org/10.1006/jagm.1993.1003 (see page 16).

[Dro96] Maciej Drozdowski. **Scheduling multiprocessor tasks — An overview**. *European Journal of Operational Research* 94:2 (1996), 215–230. ISSN: 0377-2217. URL: http://www.sciencedirect.com/science/article/pii/0377221796001233 (see pages 10, 11).

[DST97] Paolo Dell'Olmo, Maria Grazia Speranza, and Zsolt Tuza. **Efficiency and effectiveness of normal schedules on three dedicated processors**. *Discret. Math.* 164:1-3 (1997), 67–79. URL: https://doi.org/10.1016/S0012-365X(97)84781-4 (see page 10).

[EH15] Thomas Erlebach and Michael Hoffmann. **Query-Competitive Algorithms for Computing with Uncertainty**. *Bulletin of the EATCS* 116 (2015). URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/335 (see page 14).

[FMO+07] Tomás Feder, Rajeev Motwani, Liadan O'Callaghan, Chris Olston, and Rina Panigrahy. **Computing shortest paths with uncertainty**. *J. Algorithms* 62:1 (2007), 1–18. URL: https://doi.org/10.1016/j.jalgor.2004.07.005 (see page 14).

[FMP+03] Tomás Feder, Rajeev Motwani, Rina Panigrahy, Chris Olston, and Jennifer Widom. **Computing the Median with Uncertainty**. *SIAM J. Comput.* 32:2 (2003), 538–547. URL: https://doi.org/10.1137/S0097539701395668 (see page 14).

[FMP19] Dimitris Fotakis, Jannik Matuschke, and Orestis Papadigenopoulos. **Malleable Scheduling Beyond Identical Machines**. In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2019, September 20-22, 2019, Massachusetts Institute of Technology, Cambridge, MA, USA*. 2019, 17:1–17:14. URL: https://doi.org/10.4230/LIPIcs.APPROX-RANDOM.2019.17 (see page 10).

[FR96] Dror G. Feitelson and Larry Rudolph. **Towards Convergence in Job Schedulers for Parallel Supercomputers**. In: *Job Scheduling Strategies for Parallel Processing, IPPS'96 Workshop, Honolulu, Hawai, USA, April 16, 1996, Proceedings*. Ed. by Dror G. Feitelson and Larry Rudolph. Vol. 1162. Lecture Notes in Computer Science. Springer, 1996, 1–26. URL: https://doi.org/10.1007/BFb0022284 (see page 3).

[Ger96] Jordan Gergov. **Approximation Algorithms for Dynamic Storage Allocations**. In: *Algorithms - ESA '96, Fourth Annual European Symposium, Barcelona, Spain, September 25-27, 1996, Proceedings*. Ed. by Josep Díaz and Maria J. Serna. Vol. 1136. Lecture Notes in Computer Science. Springer, 1996, 52–61. URL: https://doi.org/10.1007/3-540-61680-2%5C_46 (see page 11).

[Ger99] Jordan Gergov. **Algorithms for Compile-Time Memory Optimization**. In: *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 17-19 January 1999, Baltimore, Maryland, USA*. 1999, 907–908. URL: http://dl.acm.org/citation.cfm?id=314500.315082 (see pages 11, 32, 34, 35).

[GGI+15] Marc Goerigk, Manoj Gupta, Jonas Ide, Anita Schöbel, and Sandeep Sen. **The robust knapsack problem with queries**. *Computers & OR* 55 (2015), 12–22. URL: https://doi.org/10.1016/j.cor.2014.09.010 (see page 14).

[GGK+19] Naveen Garg, Anupam Gupta, Amit Kumar, and Sahil Singla. **Non-Clairvoyant Precedence Constrained Scheduling**. In: *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*. Ed. by Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi. Vol. 132. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 63:1–63:14. URL: https://doi.org/10.4230/LIPIcs.ICALP.2019.63 (see pages 3, 17, 69, 77).

[GHH16] Marco E. T. Gerards, Johann L. Hurink, and Philip K. F. Hölzenspies. **A survey of offline algorithms for energy minimization under deadline constraints**. *J. Sched.* 19:1 (2016), 3–19. URL: https://doi.org/10.1007/s10951-015-0463-8 (see page 12).

[GJ79] M. R. Garey and D. S. Johnson. **Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)**. First Edition. W. H. Freeman, 1979. ISBN: 0716710455. URL: http://www.amazon.com/Computers-Intractability-NP-Completeness-Mathematical-Sciences/dp/0716710455 (see page 5).

[GLL+79] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.Rinnooy Kan. "Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey." In: *Discrete Optimization II*. Ed. by P.L. Hammer, E.L. Johnson, and B.H. Korte. Vol. 5. Annals of Discrete Mathematics. Elsevier, 1979, 287–326. URL: https://www.sciencedirect.com/science/article/pii/S016750600870356X (see page 4).

[GNS14] Gero Greiner, Tim Nonner, and Alexander Souza. **The Bell Is Ringing in Speed-Scaled Multiprocessor Scheduling**. *Theory Comput. Syst.* 54:1 (2014), 24–44. URL: https://doi.org/10.1007/s00224-013-9477-9 (see pages 12, 13, 65).

[GP19] Sreenivas Gollapudi and Debmalya Panigrahi. **Online Algorithms for Rent-Or-Buy with Expert Advice**. In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019,

2319–2327. URL: http://proceedings.mlr.press/v97/gollapudi19a.html (see page 17).

[GSS11] Manoj Gupta, Yogish Sabharwal, and Sandeep Sen. **The update complexity of selection and related problems**. In: *IARCS FSTTCS 2011*. 2011, 325–338. URL: https://doi.org/10.4230/LIPIcs.FSTTCS.2011.325 (see page 14).

[HEK+08] Michael Hoffmann, Thomas Erlebach, Danny Krizanc, Matús Mihalák, and Rajeev Raman. **Computing Minimum Spanning Trees with Uncertainty**. In: *STACS 2008*. 2008, 277–288. URL: https://doi.org/10.4230/LIPIcs.STACS.2008.1358 (see page 14).

[HIK+19] Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. **Learning-Based Frequency Estimation Algorithms**. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=r1lohoCqY7 (see page 17).

[HIY+16] Xin Han, Kazuo Iwama, Deshi Ye, and Guochuan Zhang. **Approximate strip packing: Revisited**. *Inf. Comput.* 249 (2016), 110–120. URL: https://doi.org/10.1016/j.ic.2016.03.010 (see page 11).

[HJP+14] Rolf Harren, Klaus Jansen, Lars Prädel, and Rob van Stee. **A (5/3 + ϵ)-approximation for strip packing**. *Comput. Geom.* 47:2 (2014), 248–267. URL: https://doi.org/10.1016/j.comgeo.2013.08.008 (see page 11).

[HSS+97] Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. **Scheduling to Minimize Average Completion Time: Off-Line and On-Line Approximation Algorithms**. *Math. Oper. Res.* 22:3 (1997), 513–544. URL: https://doi.org/10.1287/moor.22.3.513 (see page 16).

[HSW01] Han Hoogeveen, Petra Schuurman, and Gerhard J. Woeginger. **Non-Approximability Results for Scheduling Problems with Minsum Criteria**. *INFORMS J. Comput.* 13:2 (2001), 157–168. URL: https://doi.org/10.1287/ijoc.13.2.157.10520 (see page 16).

[HV96] Han Hoogeveen and Arjen P. A. Vestjens. **Optimal On-Line Algorithms for Single-Machine Scheduling**. In: *Integer Programming and Combinatorial Optimization, 5th International IPCO Conference, Vancouver, British Columbia, Canada, June 3-5, 1996, Proceedings*. Ed. by William H. Cunningham, S. Thomas McCormick, and Maurice Queyranne. Vol. 1084. Lecture Notes in Computer Science. Springer, 1996, 404–414. URL: https://doi.org/10.1007/3-540-61310-2%5C_30 (see page 16).

[HVV94] J. A. Hoogeveen, Steef L. van de Velde, and Bart Veltman. **Complexity of Scheduling Multiprocessor Tasks with Prespecified Processor Allocations**. *Discret. Appl. Math.* 55:3 (1994), 259–272. URL: https://doi.org/10.1016/0166-218X(94)90012-4 (see page 10).

[IKQ+21] Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. **Non-Clairvoyant Scheduling with Predictions**. In: *SPAA '21: 33rd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, 6-8 July, 2021*. Ed. by Kunal Agrawal and Yossi Azar. ACM, 2021, 285–294. URL: https://doi.org/10.1145/3409964.3461790 (see page 18).

[Jan02] Klaus Jansen. **Scheduling Malleable Parallel Tasks: An Asymptotic Fully Polynomial-Time Approximation Scheme**. In: *Algorithms - ESA 2002, 10th Annual European Symposium, Rome, Italy, September 17-21, 2002, Proceedings*. Ed. by Rolf H. Möhring and Rajeev Raman. Vol. 2461. Lecture Notes in Computer Science. Springer, 2002, 562–573. URL: https://doi.org/10.1007/3-540-45749-6%5C_50 (see page 10).

[JP02] Klaus Jansen and Lorant Porkolab. **Linear-Time Approximation Schemes for Scheduling Malleable Parallel Tasks**. *Algorithmica* 32:3 (2002), 507–520. URL: https://doi.org/10.1007/s00453-001-0085-8 (see page 10).

[JPS20] Zhihao Jiang, Debmalya Panigrahi, and Kevin Sun. **Online Algorithms for Weighted Paging with Predictions**. In: *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*. Ed. by Artur Czumaj, Anuj Dawar, and Emanuela Merelli. Vol. 168. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 69:1–69:18. URL: https://doi.org/10.4230/LIPIcs.ICALP.2020.69 (see page 17).

[JR19] Klaus Jansen and Malin Rau. **Improved approximation for two dimensional Strip Packing with polynomial bounded width**. *Theor. Comput. Sci.* 789 (2019), 34–49. URL: https://doi.org/10.1016/j.tcs.2019.04.002 (see page 11).

[JT10] Klaus Jansen and Ralf Thöle. **Approximation Algorithms for Scheduling Parallel Jobs**. *SIAM J. Comput.* 39:8 (2010), 3571–3615. URL: https://doi.org/10.1137/080736491 (see page 10).

[Kah91] Simon Kahan. **A Model for Data in Motion**. In: *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*. Ed. by Cris Koutsougeras and Jeffrey Scott Vitter. ACM, 1991, 267–277. URL: https://doi.org/10.1145/103418.103449 (see page 14).

[Kie88] Hal A. Kierstead. **The Linearity of First-Fit Coloring of Interval Graphs**. *SIAM J. Discret. Math.* 1:4 (1988), 526–530. URL: https://doi.org/10.1137/0401048 (see page 11).

[KR00] Claire Kenyon and Eric Rémila. **A Near-Optimal Solution to a Two-Dimensional Cutting Stock Problem**. *Math. Oper. Res.* 25:4 (2000), 645–656. URL: https://doi.org/10.1287/moor.25.4.645.12118 (see page 11).

[KR96] Claire Kenyon and Eric Rémila. **Approximate Strip Packing**. In: *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*. 1996, 31–36. URL: https://doi.org/10.1109/SFCS.1996.548461 (see page 11).

[LLM+20] Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. **Online Scheduling via Learned Weights**. In: *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*. Ed. by Shuchi Chawla. SIAM, 2020, 1859–1877. URL: https://doi.org/10.1137/1.9781611975994.114 (see page 17).

[LMH+21] Russell Lee, Jessica Maghakian, Mohammad H. Hajiesmaili, Jian Li, Ramesh K. Sitaraman, and Zhenhua Liu. **Online Peak-Aware Energy Scheduling with Untrusted Advice**. In: *e-Energy '21: The Twelfth ACM International Conference on Future Energy Systems, Virtual Event, Torino, Italy, 28 June - 2 July, 2021*. Ed. by Herman de Meer and Michela Meo. ACM, 2021, 107–123. URL: https://doi.org/10.1145/3447555.3464860 (see page 17).

[LMT+15] Giorgio Lucarelli, Fernando Machado Mendonca, Denis Trystram, and Frédéric Wagner. **Contiguity and Locality in Backfilling Scheduling**. In: *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2015, Shenzhen, China, May 4-7, 2015*. 2015, 586–595. URL: https://doi.org/10.1109/CCGrid.2015.143 (see page 12).

[LSS03] Xiwen Lu, René Sitters, and Leen Stougie. **A class of on-line scheduling algorithms to minimize total completion time**. *Oper. Res. Lett.* 31:3 (2003), 232–236. URL: https://doi.org/10.1016/S0167-6377(03)00016-6 (see page 16).

[LV18] Thodoris Lykouris and Sergei Vassilvitskii. **Competitive Caching with Machine Learned Advice**. In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, 3302–3311. URL: http://proceedings.mlr.press/v80/lykouris18a.html (see pages 6, 17).

[Mit20] Michael Mitzenmacher. **Scheduling with Predictions and the Price of Misprediction**. In: *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*. Ed. by Thomas Vidick. Vol. 151. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 14:1–14:18. URL: https://doi.org/10.4230/LIPIcs.ITCS.2020.14 (see page 17).

[MMS15] Nicole Megow, Julie Meißner, and Martin Skutella. **Randomization Helps Computing a Minimum Spanning Tree under Uncertainty**. In: *Algorithms - ESA 2015*. 2015, 878–890. URL: https://doi.org/10.1007/978-3-662-48350-3_73 (see page 14).

[MPT94] Rajeev Motwani, Steven J. Phillips, and Eric Torng. **Non-Clairvoyant Scheduling**. *Theor. Comput. Sci.* 130:1 (1994), 17–47 (see pages 17, 68, 80, 83).

[MRT07] Gregory Mounie, Christophe Rapine, and Denis Trystram. **A 3/2-Approximation Algorithm for Scheduling Independent Monotonic Malleable Tasks**. *SIAM J. Comput.* 37:2 (2007), 401–412. URL: https://doi.org/10.1137/S0097539701385995 (see pages 3, 10, 20).

[MTC02] Antonio Miranda, Luz Torres, and Jianer Chen. **On the Approximability of Multiprocessor Task Scheduling Problems**. In: *Algorithms and Computation, 13th International Symposium, ISAAC 2002 Vancouver, BC, Canada, November 21-23, 2002, Proceedings*. 2002, 403–415. URL: https://doi.org/10.1007/3-540-36136-7%5C_36 (see page 10).

[MV17] Andres Muñoz Medina and Sergei Vassilvitskii. **Revenue Optimization with Approximate Bid Predictions**. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett. 2017, 1858–1866. URL: https://proceedings.neurips.cc/paper/2017/hash/884d79963bd8bc0ae9b13a1aa71add73-Abstract.html (see page 17).

[MV22] Benjamin Moseley and Shai Vardi. **The efficiency-fairness balance of Round Robin scheduling**. *Operations Research Letters* 50:1 (2022), 20–27. ISSN: 0167-6377. URL: https://www.sciencedirect.com/science/article/pii/S0167637721001619 (see pages 17, 69, 77).

[Non13] None None. **Synergistic Challenges in Data-Intensive Science and Exascale Computing. Summary report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee, March 2013** (Mar. 2013). DOI: 10.2172/1471113. URL: https://www.osti.gov/biblio/1471113 (see pages 1, 2).

[OW00] Chris Olston and Jennifer Widom. **Offering a Precision-Performance Tradeoff for Aggregation Queries over Replicated Data**. In: *VLDB 2000*. 2000, 144–155. URL: http://www.vldb.org/conf/2000/P144.pdf (see page 14).

[PSK18] Manish Purohit, Zoya Svitkina, and Ravi Kumar. **Improving Online Algorithms via ML Predictions**. In: *NeurIPS, Montréal, Canada*. 2018, 9684–9693. URL: https://proceedings.neurips.cc/paper/2018/hash/73a427badebe0e32caa2e1fc7530b7f3-Abstract.html (see pages 3, 6, 18, 68, 69, 83).

[PSW98] Cynthia A. Phillips, Clifford Stein, and Joel Wein. **Minimizing average completion time in the presence of release dates**. *Math. Program.* 82 (1998), 199–223. URL: https://doi.org/10.1007/BF01585872 (see page 16).

[Roh20] Dhruv Rohatgi. **Near-Optimal Bounds for Online Caching with Machine Learned Advice**. In: *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*. Ed. by Shuchi Chawla. SIAM, 2020, 1834–1845. URL: https://doi.org/10.1137/1.9781611975994.112 (see page 17).

[Sch96] Andreas S. Schulz. **Scheduling to Minimize Total Weighted Completion Time: Performance Guarantees of LP-Based Heuristics and Lower Bounds**. In: *Integer Programming and Combinatorial Optimization, 5th International IPCO Conference, Vancouver, British Columbia, Canada, June 3-5, 1996, Proceedings*. Ed. by William H. Cunningham, S. Thomas McCormick, and Maurice Queyranne. Vol. 1084. Lecture Notes in Computer Science. Springer, 1996, 301–315. URL: https://doi.org/10.1007/3-540-61310-2%5C_23 (see page 16).

[Smi56] Wayne E. Smith. **Various optimizers for single-stage production**. *Naval Research Logistics Quarterly* 3:1-2 (1956), 59–66. URL: https://doi.org/10.1002/nav.3800030106 (see page 15).

[Ste97] A. Steinberg. **A Strip-Packing Algorithm with Absolute Performance Bound 2**. *SIAM J. Comput.* 26:2 (1997), 401–409. URL: https://doi.org/10.1137/S0097539793255801 (see page 11).

[SW00] Martin Skutella and Gerhard J. Woeginger. **A PTAS for Minimizing the Total Weighted Completion Time on Identical Parallel Machines**. *Math. Oper. Res.* 25:1 (2000), 63–75. URL: https://doi.org/10.1287/moor.25.1.63.15212 (see page 16).

[TWY92] John Turek, Joel L. Wolf, and Philip S. Yu. **Approximate Algorithms Scheduling Parallelizable Tasks**. In: *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '92, San Diego, CA, USA, June 29 - July 1, 1992*. 1992, 323–332. URL: https://doi.org/10.1145/140901.141909 (see pages 3, 10).

[Ves97] A.P.A. Vestjens. **On-line machine scheduling**. English. PhD thesis. Mathematics and Computer Science, 1997. ISBN: 90-386-0571-4. DOI: 10.6100/IR500043 (see page 17).

[WL20] Shufan Wang and Jian Li. **Online Algorithms for Multi-shop Ski Rental with Machine Learned Predictions**. In: *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020*. Ed. by Amal El Fallah Seghrouchni, Gita Sukthankar, Bo An, and Neil Yorke-Smith. International Foundation for Autonomous Agents and Multiagent Systems, 2020, 2035–2037. URL: https://dl.acm.org/doi/abs/10.5555/3398761.3399066 (see page 17).

[WZ20] Alexander Wei and Fred Zhang. **Optimal Robustness-Consistency Trade-offs for Learning-Augmented Online Algorithms**. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/5bd844f11fa520d54fa5edec06ea2507-Abstract.html (see page 18).

[YDS95] F. Frances Yao, Alan J. Demers, and Scott Shenker. **A Scheduling Model for Reduced CPU Energy**. In: *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*. 1995, 374–382. URL: https://doi.org/10.1109/SFCS.1995.492493 (see pages 2, 3, 12–14, 48, 57, 88).