Resource Cost Aware Scheduling Problems

Rodrigo A. Carrasco

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2013

ABSTRACT

Resource Cost Aware Scheduling Problems

Rodrigo A. Carrasco

Managing the consumption of non-renewable and/or limited resources has become an important issue in many different settings. In this dissertation we explore the topic of resource cost aware scheduling. Unlike the purely scheduling problems, in the resource cost aware setting we are not only interested in a scheduling performance metric, but also the cost of the resources consumed to achieve a certain performance level.

There are several ways in which the cost of non-renewal resources can be added into a scheduling problem. Throughout this dissertation we will focus in the case where the resource consumption cost is added, as part of the objective, to a scheduling performance metric such as weighted completion time and weighted tardiness among others.

In our work we make several contributions to the problem of scheduling with non-renewable resources. For the specific setting in which only energy consumption is the important resource, our contributions are the following.

- We introduce a model that extends the previous energy cost models by allowing more general cost functions that can be job-dependent.
- We further generalize the problem by allowing arbitrary precedence constraints and release dates.
- We give approximation algorithms for minimizing an objective that is a combination of a scheduling metric, namely total weighted completion time and total weighted tardiness, and the total energy consumption cost.
- Our approximation algorithm is based on an interval-and-speed-indexed IP formulation.

We solve the linear relaxation of this IP and we use this solution to compute a schedule.

- We introduce the concept of α-speeds, which extend the α-points technique to problems with multiple speeds.
- We show that these algorithms have small constant approximation ratios.
- Through experimental analysis we show that the empirical approximation ratios are much better than the theoretical ones and that in fact the solutions are close to optimal.
- We also show empirically that the algorithm can be used in additional settings not covered by the theoretical results, such as using flow time or an online setting, with good approximation and competitiveness ratios.

Because our model considers job-dependent energy costs, we can further generalize our results to the setting where multiple resources are available, and the consumption level of all those resources will determine the speed at which jobs are processed. We call this setting *resource cost aware scheduling*. We make several contributions to the resource cost aware scheduling problem.

- We introduce a model that extends the previous cost models (linear, convex, and other energy models) by allowing a more general relation between job processing time (or equivalent processing speed) and resource consumption.
- We further generalize the problem by allowing arbitrary precedence constraints and release dates.
- We give approximation algorithms for minimizing an objective that is a combination of a scheduling metric (weighted completion time) and resource consumption cost.

We consider a more general model of resource cost than has previously been used. The resource dependent job processing time literature either focuses on job's processing times that depend linearly on resource consumption or a convex relation of the form $(\rho_i/u_i)^k$, generally

considering only a single resource. Our setting captures both of these models by considering an arbitrary non-negative speed function $\mathcal{S}(\mathbf{\Psi}^{(i)})$, where $\mathbf{\Psi}^{(i)} \in \mathbf{\Psi} = \{\mathbf{\Psi}^{(1)}, \ldots, \mathbf{\Psi}^{(q)}\}$ denotes one of the q allowable operating points of the resources. We also generalize the resource cost, which is generally linear in the literature, by considering an arbitrary non-negative job-dependent resource cost function $\mathcal{R}_i(\mathbf{\Psi}^{(i)})$.

We state here the most general of our results.

Theorem 0.1. *Given n jobs with precedence constraints and release dates and a general non-negative resource cost function, there is an $O(1)$-approximation algorithm for the problem of non-preemptively minimizing a weighted sum of the completion time and resource cost.*

The constants in the $O(1)$ are modest. Given some $\epsilon > 0$, the algorithm has a $(4+\epsilon)$-approximation ratio when only precedence constraints exist, and $(3+2\sqrt{2}+\epsilon)$-approximation ratio when release dates are added.

Because some of our algorithms use resource augmentation to deal with nonlinear scheduling performance metrics (like weighted tardiness), we can further extend the use of our algorithms to the setting where no resource cost is considered but a general convex non-decreasing scheduling performance metric is used. We make several contributions to the problem of scheduling jobs with non-decreasing convex cost functions:

- We introduce a model that extends the previous models by allowing a more general non-linear job-dependent function of the completion time as the scheduling metric.
- We propose a new approximation algorithm for minimizing the total cost, with arbitrary precedence constraints.
- Our algorithm builds on both the α-point and resource augmentation techniques. We show that our algorithm has a small constant approximation ratio and a small speed-scaling ratio for several important scheduling metrics, namely the total weighted tardiness, the total weighted tardiness squared, and the total completion time squared. The results of our numerical experiments show that the practical performance of our algorithm is significantly superior to the theoretical bounds.

- We compare the performance of our algorithm with other available methods for the total weighted tardiness problem by using the test instances from the OR Library [Beasley, 1990]. We show that our algorithm is capable of computing approximate optimal solutions for all the available test problems, even those with $n = 100$ jobs. Thus, we are able to establish lower bounds on the optimal solutions for instances where the optimal schedule is currently not known. Our algorithm takes less than a second to solve even the larger instances (with $n = 100$ jobs), which is at least one order of magnitude faster than current methods. Furthermore, we show that on average only a 2% speed-up is required to achieve the best known result; and, in fact, in several cases no speed-up factor is required.

Our main result can be summarized in the following theorem

Theorem 0.2. *Given n jobs with arbitrary precedence constraints and convex non-decreasing cost functions $f_i(C_i)$ for each job i, there is a $O(1)$-speed 1-approximation algorithm for the problem of minimizing the total non-linear cost $\sum f_i(C_i)$.*

The speed scaling constant is relatively small. Given $\epsilon > 0$ our algorithm is a $(4+\epsilon)$-speed 1-approximation algorithm.

Finally, we also consider the energy aware scheduling problem in which the size of a job is only known after the machine finishes processing it, but only the size probability distribution is known in advance. This is a common setting in CPUs. We also make several contributions to this particular setting.

- We propose a dynamic programming formulation which is optimal in expectation.
- We compute the optimal speeds required, given any state in the dynamic programming recursion.
- We present a policy for the case where the completion time is the scheduling performance metric, and show that this policy is optimal when only two possible sizes exist.

- We leave as an open conjecture that our policy is also optimal for the case with arbitrary job sizes, since we are only able to show through simulations that this is true in all the tested instances.

This page is intentionally printed only with this statement.

Table of Contents

This page is intentionally printed only with this statement.

List of Figures

List of Tables

This page is intentionally printed only with this statement.

List of Algorithms

This page is intentionally printed only with this statement.

Acknowledgments

It would be impossible to be brief and at the same time correctly acknowledge everyone who, in one way or another, supported me during my PhD, hence, I won't be. The past few years have been an amazing experience of academic, professional, and personal growth, which would not have been as great if it wasn't for so many people that were there for me along the way.

I'll start by thanking my two advisors, Garud Iyengar and Cliff Stein. The results in this dissertation were possible thanks to their support, knowledge, and guidance. The discussions in our weekly meetings were oftentimes challenging but extremely rich and I was able to learn a lot from our interactions. Both have been great role models and have taught me very much, not just research-wise, but professionally as well. I consider myself lucky for having them both as advisors and friends. I also want to thank them the opportunity they gave me to teach the *Asset Allocation* course. It was a great experience that made me realize how much I enjoy teaching.

Into the specifics of this dissertation, I am very grateful to my dissertation committee: Daniel Bienstock and Vineet Goyal, both from Columbia University, and Viswanath Nagarajan from IBM at the T.J. Watson Research Center. Their comments and the discussions I had with them prior to the defense helped me improve this document. I appreciate their time and commitment.

I would also like to thank all the professors at the Industrial Engineering and Operations Research Department. I was very lucky to have interactions on a regular basis with most of them and they were of great support along the way. In particular I would like to thank Daniel Bienstock, Maria Chudnovsky, Martin Haugh, Donald Goldfarb, Ciamac Moallemi,

Katya Scheinberg, Jay Sethuraman, Ward Whitt, David Yao, and again Garud Iyengar and Cliff Stein. Their courses were really good and I thank them for their time and dedication. Although I did not have them as professors, I would also like to thank José Blanchet, Emanuel Derman, Guillermo Gallego, Soulaymane Kachani, Peter Norden, Lucius Riccio, and Karl Sigman. Our random conversations about anything and everything made this place even more enjoyable.

My dissertation has benefited greatly with the discussions with several other researchers I would like to thank: Ioannis Akrotirianakis and Amit Chakraborty from Siemens Corporate Research; Stephen Boyd, from Stanford University; Tolga Cezik, from Amazon Inc.; Nicole Megow, from TU Berlin; Kirk Pruhs, from University of Pittsburgh; and David Shmoys, from Cornell University. I would also like to thank all the anonymous reviewers of our papers. They gave us valuable feedback to improve our work.

IEOR's staff is unparalleled. They keep the department running and work very hard to make sure we have a family-like atmosphere. Their help and wonderful conversations of things that, thankfully, had nothing to do with work will be always cherished. In particular I would like to thank Jenny Mak, who was always supportive of all the crazy ideas and activities I came up with; Jaya Mohanty and Shi Yee Lee, that made sure we were paid or reimbursed on time (and that there were cookies in the lounge); and Adina Berríos, María Casuscelli, Ufei Chan, Risa Cho, Donella Crosgnach, Jessica Gray, Samuel Lee, Mindy Levinson, Cynthia Malave-Baez, Michael Mostow, Carmen Ng, and Darbi Roberts, who were always organizing things for us and made sure we were OK.

When you spend a significant amount of your time working and studying, your classmates and acquaintances soon become part of your extended family. In this sense I was very lucky to meet so many wonderful strangers that today I call friends. First, my officemates of the great 313A, with whom I spent so many hours and made this experience more enjoyable. Special thanks go to them: Andrew Ahn, Tulia Herrera, Jinbeom Kim, Arseniy Kukanov, Tony Qin, Yixi Shi, Xingbo Xu, Cecilia Zenteno, Haowen Zhong, and those that left or arrived to 313A during my time: Serhat Aybat, Nur Ayvaz, Erez Cohen, Antoine Desir, Gonzalo Muñoz,

Fahad Saleh, Rishi Talreja, and Chun Ye. Also to all the members of Garud's research group: Carlos Abad, Serhat Aybat, Yupeng Chen, Chen Chen, Daniel Guetta, Suraj Keshri, and Rhea Qiu. Their comments during our biweekly meetings were of great help to improve my work. I would also like to thank all the other friends I met along the way as well as my NYC family, whose support was invaluable: Eyjólfur Ásgeirsson, Edson Bastos, Berk Birand, Xinyun Chen, Lizzie Cruz, Romain Deguest, Jing Dong, Juan Elias, Itai Feigenbaum, Magdalena Gil, Olivia Gillham, Vladimir Glasinovic, Denise Hauva, Rouba Ibrahim, Andrew Kang, Song-Hee Kim, Olga Kolesnikova, Thiam Hui Lee, James Lenzi, Juan Li, Yunan Liu, Yina Lu, Shiqian Ma, Peter Maceli, Shyno Mathew, Alex Michalka, Amal Moussa, Gordon Pang, Antonella Pavese, Radka Pickova, Matthieu Plumettaz, Xianhua Peng, David Phillips, Verónica Rodríguez, Johannes Ruf, Ali Sadighian, Marco Santoli, Gustavo Schmidt, Steen Schmidt, Marilin Sierra, Irene Song, Jingjing Song, Shyam Sundar, Xiaocheng Tang, Fanny Thomas, Johanna Urzedowski, Josh Van Gundy, Abhinav Verma, Aya Wallwater, Burcu Yildirim, John Zheng, and Yori Zwols. All of them, in one way or another, made the time fly, the good experiences much more enjoyable, and the hard times more bearable. I sincerely hope I'm not missing any names.

My family and friends back home deserve special prizing. Not only they have been stuck with me for many years, which in itself deserves a mention, but not even the distance has kept them from being interested in what I was doing. Their support was invaluable. My deepest gratitude goes to my parents, Rodrigo and Loreto, who have always showed so much interest in my work and in our wellbeing. They were the ones that fostered my interest in science and mathematics, and their support and encouragement through all my life has lead me to where I am right now. There are not enough words of gratitude towards them for all they have done for me. I would also like to thank my parents-in-law, Max and Cecilia. They have gone above and beyond to support and help me through all these years and their interest and encouragement has been a blessing. Also their children and their spouses, Alejandro and María José; Verónica and Rodrigo; Marcela and Javier; and Ricardo, have been really important. They are the brothers and sisters I never had and I feel incredibly lucky for having

them in my life. I would also like to thank my long-time friends Rolando Dünner, Tomás Ibáñez, Peter Leatherbee, Rolando Ruiz, and Sergio Zacharías for their encouragement and all the great times we had whenever we had the chance to get together.

Such an important section has to have a fitting ending, and I cannot think of anything more perfect than thanking the most important person in my life: my wonderful wife Daniela. Daniela had the really hard job leaving everything behind and accompany me in this journey, with much more uncertainty than I had. Yet, she found the strength to get a Masters degree and a Doctorate in Occupational Therapy, work a full-time job, help me more than I could fathom, and give birth and rise a wonderful little (well, really not so little) girl. I am left without words to thank her for all her support during all the hard times in this process, all the enjoyment she added to the good times, specially during our camping trips to California and Utah and all the adventures we had, and the help she gave me every single day. I don't know anyone else that can lit a room like she does and I feel myself blessed of meeting her and having her as my partner and best friend.

To finish, in the words of my youngest coauthor, my daughter Alicia: *80.v hb bv 0b 0 kbv 0oiljnuji k.00 m nngvhb X nb b 3cxzS* .

This research was partially supported by NSF grants CCF-0728733 and CCF-0915681; NSF grant DMS-1016571, ONR grant N000140310514, and DOE grants DEFG02-08ER25856 and DE-AR0000235; and Fulbright/Conicyt Chile.

This work is dedicated to Daniela and Alicia.

This page is intentionally printed only with this statement.

Chapter 1

Introduction

MANAGING non-renewable resources has become an important issue in many different settings. In this dissertation we explore the topic of resource cost aware scheduling, where the consumption cost of non-renewable resources is combined with a scheduling problem. We present several different models and settings, and we develop and discuss both approximation and optimization algorithms for them. In Chapter 2, we give an overview of the state-of-the-art in this topic, including a literature review on resource dependent job processing times. This topic is directly related to the resource and energy aware scheduling problems, but was developed independently.

In Chapter 3, we formulate the basic energy aware scheduling problem that we will further develop and extend throughout this work. In this setting only energy consumption determines the speed at which jobs are processed and the objective is to minimize the sum of the total energy consumption cost and some scheduling performance metric. In Chapter 3 we make several contributions to the problem of scheduling with non-renewable resources:

- We introduce a model that extends the previous energy cost models by allowing more general cost functions that can be job-dependent.

- We further generalize the problem by allowing arbitrary precedence constraints and release dates.

- We give approximation algorithms for minimizing an objective that is a combination of a scheduling metric, namely total weighted completion time and total weighted tardiness, and the total energy consumption cost.
- Our approximation algorithm is based on an interval-and-speed-indexed IP formulation. We solve the linear relaxation of this IP and we use this solution to compute a schedule.
- We introduce the concept of α-speeds, which extend the α-points technique to problems with multiple speeds.
- We show that these algorithms have small constant approximation ratios.

In Chapter 4, we analyze the performance of the proposed algorithms through experimental analysis. We show that, although the theoretical worst-case scenario bounds of our approximation algorithms are small, they perform much better and in fact the algorithm's output is a schedule whose value is very close to optimal. Additionally, in Chapter 4 we present several heuristic improvements and extensions of our algorithms. These extensions effectively reduce the empirical approximation ratios. In Chapter 4 we also show that we can apply our algorithms in different settings like online, multiple machines, and using total weighted flow time as scheduling metric, with good results.

In Chapter 5 we further extend our algorithms to the resource cost aware scheduling problem, where the speed at which jobs run is determined by an arbitrary number of resources. The problem of managing multiple resources to control the performance of the resulting schedule arises in many industrial applications. We make several contributions to this general setting:

- We introduce a model that extends the previous cost models (linear, convex, and other energy models) by allowing a more general relation between job processing time (or equivalently processing speed) and resource consumption.
- We give approximation algorithms for minimizing an objective that is a combination of a scheduling metric (weighted completion time) and resource consumption cost.

- We show that these algorithms have the same small constant approximation ratios shown in the algorithms of Chapter 3.

We consider a more general model of resource cost than has previously been used. As noted before, the resource dependent job processing time literature either focuses on job's processing times that depend linearly on resource consumption or a convex relation of the form $(\rho_i/u_i)^k$, generally considering only a single resource. Our setting captures both of these models by considering an arbitrary non-negative speed function $\mathcal{S}(\mathbf{\Psi}^{(i)})$, where $\mathbf{\Psi}^{(i)} \in \mathbf{\Psi} = \{\mathbf{\Psi}^{(1)}, \ldots, \mathbf{\Psi}^{(q)}\}$ denotes one of the q allowable operating points of the resources. This is a very common setting in many industrial applications where resources only have a discrete number of possibilities. We also generalize the resource cost, which is generally linear in the literature, by considering an arbitrary non-negative job-dependent resource cost function $\mathcal{R}_i(\mathbf{\Psi}^{(i)})$.

We state here the most general result of this Chapter

Theorem 1.1. *Given n jobs with precedence constraints and release dates and a general non-negative resource cost function, there is an $O(1)$-approximation algorithm for the problem of non-preemptively minimizing a weighted sum of the completion time and resource cost.*

The constants in the $O(1)$ are modest. Given some $\epsilon > 0$, the algorithm has a $(4+\epsilon)$-approximation ratio when only precedence constraints exist, and $(3+2\sqrt{2}+\epsilon)$-approximation ratio when release dates are added.

We extend the interval-indexed IP proposed in Chapter 3 to handle resource costs and speed scaling, and then design a new α-point based rounding algorithm to obtain the resulting schedules. We assume first that we have a discrete set of q allowable resource operating points $\mathbf{\Psi} = \{\mathbf{\Psi}^{(1)}, \ldots, \mathbf{\Psi}^{(q)}\}$, and that the speed at which the job is precessed is a general non-negative function of the resource operating point. In our interval-indexed IP, a variable x_{ijt} is 1 if job i runs at resource operating point $\mathbf{\Psi}^{(j)}$ and completes in interval t. We can then extend the standard interval-indexed integer programming formulation to take the extra dimensions of resource consumption and speed into account. Once we have solved its linear

program relaxation (which we denote LPi), we need to determine *both an α-point and α-speed*. The key insight is that by "summarizing" each dimension appropriately, we are able to make the correct choice for the other dimension. At a high level, we first choose the α-point by "collapsing" all pieces of a job that completes in the LPi in interval t (these pieces have different speeds), being especially careful with the last interval, where we may have to choose only some of the speeds. We then use *only* the pieces of the job that complete before the α-point to choose the speed, where the speed is chosen by collapsing the time dimension and then interpreting the result as a probability mass function (pmf), where the probability that the job is run at operating point $\mathbf{\Psi}^{(j)}$ depends on the total amount of processing done at that operating point. We then apply the concept of α-speeds defined in Chapter 3, which is related to the expected value under this pmf, and run the job at this speed. We combine this rounding method with extensions of the more traditional methods for dealing with precedence constraints and release dates to obtain our algorithm.

In Chapter 6 we further explore the energy aware scheduling problem by considering the setting where job sizes are only known after the job is finished. This setting is very common when processing jobs in CPUs, where the machine does not know in advance the amount of time or machine cycles required to finish a task. In this setting we will only assume that job sizes have a known discrete probability distribution and we propose an algorithm that determines a scheduling rule that is optimal in expectation.

In Chapter 7 we present an approximation algorithm for the setting where only scheduling performance is of interest, but we allow for a very general cost function on the completion time of each job. This algorithm is a direct result of the techniques discussed in Chapter 3, but because of its generality and importance as a scheduling application we further analyze this setting, presenting both theoretical and experimental results. We make several contributions to the problem of scheduling jobs with non-decreasing convex cost functions:

- We introduce an IP based model that extends previous models by allowing a more general non-linear job-dependent function of the completion time as the scheduling metric.

- We propose a new approximation algorithm for minimizing the total cost, with arbitrary precedence constraints.

- Our algorithm builds on both the α-point and resource augmentation techniques. We show that our algorithm has a small constant approximation ratio and a small speed-scaling ratio for several important scheduling metrics, namely the total weighted tardiness, the total weighted tardiness squared, and the total completion time squared. The results of our numerical experiments show that the practical performance of our algorithm is significantly superior to the theoretical bounds.

- We compare the performance of our algorithm with other available methods for the total weighted tardiness problem by using the test instances from the OR Library [Beasley, 1990]. We show that our algorithm is capable of computing approximate optimal solutions for all the available test problems, even those with $n = 100$ jobs. Thus, we are able to establish lower bounds on the optimal solutions for instances where the optimal schedule is currently not known. Our algorithm takes less than a second to solve even the larger instances. This is at least one order of magnitude faster than current methods. Furthermore, we show that on average only a 2% speed-up is required to achieve the best known result; and, in fact, in several a speed-up factor of 0 was enough to achieve the best known value.

Our main result can be summarized in the following theorem

Theorem 1.2. *Given n jobs with arbitrary precedence constraints and convex non-decreasing cost functions $f_i(C_i)$ for each job i, there is a $O(1)$-speed 1-approximation algorithm for the problem of minimizing the total non-linear cost $\sum f_i(C_i)$.*

Finally, in Chapter 8 we discuss some of the open questions and future research lines that our work leaves.

This page is intentionally printed only with this statement.

Chapter 2

Preliminaries

Non-renewable resource consumption management is fast emerging as a problem of critical importance. There is always a trade-off between resource consumption and performance: more resource consumption typically results in better performance. This trade-off also arises in many scheduling problems, where resource management decisions must be combined with the scheduling decisions to optimize a global objective.

Recently, scheduling problems in which one has to trade scheduling performance, using metrics such as weighted completion time, weighted tardiness, or flow time, with CPU processing speed, and therefore the energy consumed, have been extensively studied. However, the problem of balancing resource consumption with scheduling performance was proposed much earlier. Vickson [Vickson, 1980] observed that in many practical settings, the processing time of a job depends on the amount of resources (e.g. catalizer, workforce size, energy, etc.) utilized, and the relationship between resource utilization and processing time depends on each job's characteristics. Other examples of scheduling problems with resource dependent job processing time include repair and maintenance processes [Duffuaa *et al.*, 1999]; ingot preheating processes in steel mills, where the batches need to be scheduled and the amount of gas used and the concentration level determine the time required to preheat the ingots [Janiak, 1991; Williams, 1985]; many workforce intensive operations; VLSI circuit design [Monma *et al.*, 1990]; and more recently processing tasks in a CPU, where the job processing times

depends on CPU speed, the available RAM, bus speed, as well as other system resources.

For convenience, throughout this work we will use an extended version of the three part notation of Graham et al. [Graham *et al.*, 1979] to refer to the different scheduling problems that we will address. The notation is of the form $\alpha|\beta|\gamma$, where α indicates the machine setting: 1 for one machine, P for multiple parallel machines, etc.; β indicates the problem constraints: r_j for release dates, $prec$ for precedence constraints, or $pmnt$ for preemption; and γ indicates the performance metric. For example, $1|r_i, prec|\sum \mathcal{E}_i(s_i) + w_iC_i$, will refer to the problem setting with 1 machine, with r_i release dates, precedence constraints, and the weighted completion time as the scheduling performance metric, together with the total energy cost. Similarly, the $1|r_i, prec|\sum \mathcal{E}_i(s_i) + w_iT_i$ will refer to the same setting, but with tardiness as the scheduling performance metric. In both of them the term $\mathcal{E}_i(s_i)$ indicates that the energy cost is also added as a performance metric.

Before delving into the literature review there are several concepts related to algorithm design that should be clarified. Because many of the problems we will address are NP-hard, there are no efficient algorithms to solve them unless P = NP. We are still interested in saying something about these NP-Hard problems, since there are many related real-life applications, so the fact that no efficient algorithms exist for these NP-Hard problems will leave us with three main possibilities, all of which will be addressed in this dissertation. First, we can develop non-polynomial algorithms and only solve very small instances. In Chapter 3 we will present an IP formulation with which we can compute the exact solution for the Energy Aware Scheduling problem. We will also show, in Chapter 4, that we are limited to solve cases with up to 10 jobs with this exact formulation. Bigger instances will require too much time and resources to be solved.

A second option to deal with NP-Hard problems is to find special cases that can be solved efficiently. In Chapter 3 we will show that for certain special cases we can solve the Energy Aware Scheduling problem in polynomial time, by using an extension of Smith's Rule.

The third option to deal with NP-Hard problems, is to compute approximate optimal solutions. Most of our work will be in this third option: we will present several approxi-

mation algorithms for different resource cost scheduling problems. The main idea behind approximation algorithms is that, by using a polynomial time algorithm, we can compute a solution that is at most ρ times the optimal solution for any instance of the problem, i.e. $ALG \leq \rho \cdot OPT$. Algorithms with this property are called a ρ-approximation algorithm, and ρ is called the approximation ratio, which could depend on the instance parameters. In the online setting, because we don't have all the future information available, we will use a slightly different definition: *competitiveness*. A ρ-competitive algorithm, is an algorithm that can compute, in polynomial time, a solution that is at most ρ times the optimal offline solution, i.e. the solution that has all the future information for each instance. A very good reference on algorithms in general is [Cormen *et al.*, 2001], which includes a discussion on approximation algorithms in chapter 35. For approximation algorithms, [Williamson and Shmoys, 2011] isan excellent resource for techniques and analysis.

One last concept we will use regarding algorithms is *resource augmentation*. The main drawback in the analysis of approximation algorithms is that the approximation ratio has to consider the worst case scenario, which might never or rarely occur in real-life applications. As a way circumvent this limitation Kalyanasundaram and Pruhs [Kalyanasundaram and Pruhs, 2000] proposed analyzing an approximation algorithm by considering additional resources. Although the first examples of resource augmentation analysis are much older [Sleator and Tarjan, 1985], this technique has become popular only recently. The main idea in the analysis is that we compare the optimal solution to the output of our algorithm considering that we have additional resources (like faster machines, additional machines, etc.). We will call an algorithms an s-speed ρ-approximation algorithm, if the output of the algorithm is at most ρ times the optimal solution, but considering that our algorithm runs on a machine that is s times faster.

2.1 Resource Dependent Job Processing Time

The literature on *Resource Dependent Job Processing Time* (RDJ) problems has mainly focused on two models. In the first model the processing time p_i of job i as function of resource consumption level u_i is piece-wise linear function of the form $p_i(u_i) = \min\{\underline{p}_i, b_i - a_i u_i\}$, where a_i, b_i are job parameters and $\underline{p}_i$ is the smallest possible processing time [Cheng *et al.*, 1998; Cheng *et al.*, 2001; Daniels, 1990; Daniels and Sarin, 1989; Janiak, 1987; Janiak, 1991; Janiak and Kovalyov, 1996; Van Wassenhove and Baker, 1982; Vickson, 1980; Xu *et al.*, 2011]. In more recent work, the processing time as a function of the resource consumption level is assumed to be a decreasing, convex function of the form $p_i(u_i) = (\rho_i/u_i)^k$ for some $\rho_i > 0$ and $k > 0$ [Kaspi and Shabtay, 2006; Shabtay and Kaspi, 2004; Shabtay and Steiner, 2011; Wang and Wang, 2011]. The primary justification for this model is that it captures the decreasing marginal improvements that is observed in practical applications [Shabtay and Kaspi, 2004].

The trade-off between resource consumption and performance is modelled in several different ways. In [Cheng *et al.*, 1998; Cheng *et al.*, 2001; Daniels, 1990; Daniels and Sarin, 1989; Kaspi and Shabtay, 2006; Shabtay and Steiner, 2011; Van Wassenhove and Baker, 1982] the authors consider a bicriteria approach where the objective is to reduce resource consumption, and simultaneously optimize the scheduling metric. On the other hand, [Janiak, 1987; Janiak and Kovalyov, 1996; Shabtay and Kaspi, 2004; Wang and Wang, 2011; Xu *et al.*, 2011] optimize the scheduling performance for given bound or budget on the available resources or vice versa, i.e. they optimize the resource consumption given a bound in the scheduling performance. A survey of the many of the different approaches to these problems can be found in [Shabtay and Steiner, 2007]. The work we present in Chapter 5 can be considered as a generalization of the models used so far in the RDJ literature, using the weighted completion time as the schedule performance metric.

2.2 Energy Aware Scheduling

Energy Aware Scheduling (EAS) is a relatively new area within the scheduling literature and a very important example of resource aware scheduling problems that has recently received much attention [Albers, 2010; Andrew *et al.*, 2010; Atkins *et al.*, 2011; Bansal *et al.*, 2008; Bansal *et al.*, 2010; Bansal and Pruhs, 2005; Chen *et al.*, 2005; Irani *et al.*, 2007; Jawor, 2005; Kwon and Kim, 2005; Yun and Kim, 2003]. Interestingly, this setting has been studied completely separate from the RDJ setting, although EAS can be interpreted as a subproblem of RDJ, with a single resource (energy) and a specific energy cost function. In this setting the objective is to take into consideration the energy cost associated to the speed at which the jobs are processed.

This setting has become very relevant given the energy consumption of the now massive data centres that companies are using. To give a sense of how much energy we are talking about, consider Google, which states that the servers in its datacenter, which they claim are much more efficient than the average industry server, consume 1 kJ per query on average [Google, 2009]. In May 2013, just in the US, users made in average more than 440 million queries per day using Google's search engine [Comscore, 2013]. This amounts to consuming 44.2 million kWh per year, equivalent to more than $6,000$ average NY households [DOE, 2011]. Additionally, CPUs account for 50-60% of a typical server's energy consumption [Albers, 2009]; consequently, CPU energy management is especially important. Energy management is also important for smaller devices such as laptops and other mobile devices, since their resources are limited. It is clear that when scheduling computing tasks, it is important to take both the relevant scheduling quality of service (QoS) metrics such as makespan, weighted completion time or weighted flow time, and the energy consumption into account.

There are many ways in which energy consumption can be introduced into a scheduling problem, depending on the characteristics of the machine and the jobs that are being processed. There are two main areas of research in EAS problems which I'll further describe next: *power down* and *speed-scaling* settings.

2.2.1 Power Down Setting

The power down setting, as its name suggests, is the setting in which we are interested in determining when to power down a machine within an idle period. Every time the machine is powered down there is a fixed cost to power it back on, which accounts for the additional energy required to power up the machine later. We do not address this problem in any of our models but we list some of the most important references in this subsection for completeness.

The general setting is as follows. We are given a machine that, at any point in time, can be in any of m different states; one active state and $m-1$ *idle* ones. Associated to each state s_i is an energy consumption rate r_i. Without loss of generality we can assume that $r_1 > r_2 > \ldots > r_m$, and thus the active state will be state S_1. Additionally, there is a cost a_i for *activating* the machine when in idle state i, that is going from state s_i to the active state s_1. The problem is to decide, within every idle period, in which state should the machine be, given that it must be in the active state once a new job arrives (i.e. the idle period ends). Note that it does not make sense to delay jobs to allow a larger idle period, and thus go to lower idle states, since we will end up consuming more energy, plus we will worsen the scheduling performance. Hence, we are only interested in determining the state in which the machine will be within an idle period.

This problem must be considered in an on-line setting, where the length of the idle period is not known. If the idle period's length was known, then the state in which to leave the machine is easily computed.

Although this problem can be traced back to Hwang et al. [Hwang and Wu, 1997], and Srivastava et al. [Srivastava *et al.*, 1996], who introduced the problem in a predictive setting, the simplest case, when only two states are available (s_1 and s_2) and nothing is known about the idle period's length, is even older. This setting is nothing more than another take on the *ski rental problem*, a basic problem in online algorithms. Benini et al. in [Benini *et al.*, 2000] give a comprehensive survey on many of the techniques used in several different strategies for the power down problem, including predictive strategies, stochastic optimal control, adaptive learning, etc.

Algorithm 2.1 LOWER ENVELOPE

1 At any time t in an idle period, set the machine to state k where $k = \arg\min_i\{r_i t + a_i\}$.

For the simple case with only two states and $r_1 = 1$ and $r_2 = 0$, the best deterministic strategy is to wait in the active state for $a_2 = a$ units of time after an idle period begins, and then change to s_2. It is easy to prove that this is a 2-competitive algorithm, and in fact it is the best a deterministic algorithm can do. One can do better with randomized algorithms, where a probability density function $p(t)$ defines the probability that the systems powers down to the idle state within t units of the idle period. In that case Karlin et al. [Karlin *et al.*, 1994] showed that a $\frac{e}{e-1} \approx 1.58$ competitive ratio is achievable in average, when $p(t) = \frac{1}{(e-1)^a} e^{\frac{t}{a}}$ for $t \in [0, a]$ and $p(t) = 0$ otherwise. Karlin's results are in the context of the *ski rental problem* but are directly applicable to the power-down problem.

Back to the general case, Irani, Shukla, and Gupta in [Irani *et al.*, 2003] showed that when no knowledge about the idle period exists, the best competitive ratio a deterministic algorithm can achieve is 2. They also give a simple algorithm that achieves this ratio, known as LOWER ENVELOPE, which is based on the observation that if the length of the idle period was known to be T, the optimal strategy is to set the machine to state s_{i^*} where $i^* = \arg\min_i\{r_i T + a_i\}$. Algorithm 2.1 details the Lower Envelope algorithm.

The ratio can be further improved if the probability distribution of the idle period's length $q(t)$ is known. Irani et al. [Irani *et al.*, 2003] device an algorithm by first observing that if only two states s_{i-1} and s_i exist, a deterministic algorithm that powers down at time t within the idle period will have an expected energy consumption given by

$$\mathbb{E}[E(t)] = \int_0^t (r_{i-1}T)q(T)dT + \int_t^\infty (r_{i-1}T + r_i(T-t) + a_i - a_{i-1})q(T)dT. \tag{2.1}$$

An offline algorithm will choose $t_j = \frac{a_{j-1} - a_j}{r_j - r_{j-1}}$ to minimize this quantity. Similarly, the online version of this algorithm which they call PROBABILITY-BASED LOWER ENVELOPE will use these breakpoints t_j to define the state in which the machine is within the idle period. Algorithm 2.2 details this modified online algorithm.

The results in [Irani *et al.*, 2003] have been further extended by Augustine et al. [Augustine *et al.*, 2004], to incorporate arbitrary transition values among states. They show that a generalization of Probability-Based Lower Envelope results in a $3+2\sqrt{2}$-competitive algorithm. They go even further by proposing a general algorithm for any system that is $(\rho^* + \epsilon)$-competitive, where ρ^* is the best competitiveness for that system, and the solution can be computed in $O(m^2 \log m \log(\frac{1}{\epsilon}))$, where m is the number of states in the machine. The algorithm is much more complicated and their results are based on the fact that for any system there is a strategy that is ρ-eager (which means that it achieves exactly a ρ-competitiveness for every instance) and then they develop the algorithm using this result.

The surveys by Albers [Albers, 2009] and by Irani and Pruhs [Irani and Pruhs, 2005] contain further algorithms and settings additional to the main ones described here.

2.2.2 Speed Scaling Setting

The other main setting in which energy consumption is incorporated within the scheduling problem is though *speed scaling*. Most modern CPUs allow users to dynamically change the speed at which the processor runs; the lower the speed, the less energy used, and the relationship is device-dependent but typically superlinear. Thus, the energy consumed can be controlled by speed scaling. At the same time a machine running at a lower speed will time more time to process a certain job, hence a reduction in energy consumption will generally produce a detriment in the scheduling metric's performance.

Generally, the power P consumed is a polynomial function of speed s of the form $P(s) = s^\beta$ for some constant $\beta \in [2, 3]$ and $v \geq 0$. Thus, the total energy consumed in processing a job that requires ρ cycles and runs at speed s will be given by $E(s) = v\rho s^{\alpha-1}$. Recent work uses a more general power function with minimum regularity conditions, like non-negativity,

Algorithm 2.2 Probability-Based Lower Envelope

1 At any time t in an idle period, set the machine to state k where $k = \arg\max_i\{t_i : t_i < t\}$.

but in all the cases the power function is *not* job-dependent since the jobs are homogeneous [Andrew *et al.*, 2009; Bansal *et al.*, 2009].

Researchers have focused on three challenging speed scaling settings: energy minimization and deadlines, scheduling with an energy or QoS budget, and balancing energy consumption and scheduling performance. For all the mentioned cases both offline and online settings have been considered in the literature. See the surveys by Albers [Albers, 2009], Irani and Pruhs [Irani and Pruhs, 2005] for details in some of them.

2.2.2.1 Energy Minimization and Deadlines

Historically, the first researchers to study the speed scaling were Yao et al. [Yao *et al.*, 1995]. In this setting we are given n jobs, where each job i has a release date r_i, a processing requirement in CPU cycles ρ_i, and a deadline d_i. The objective is to find a preemptive schedule and the speed s_i at which each job J_i runs such that no job misses its deadline and the energy consumption is minimized. Clearly, for arbitrary instances this is only achievable if the CPU speed is not bounded.

Let D_I define the set of jobs that must be processed in interval $I = [t_1, t_2]$, i.e. $D_I = \{J_i : [r_i, d_i] \subseteq I\}$, then the density of this interval is given by

$$\Delta_I = \frac{1}{|I|} \sum_{i \in D_I} \rho_i. \tag{2.2}$$

In [Yao *et al.*, 1995] Yao et al. they use these densities to construct an algorithm known as YDS, described in Algorithm 2.3, and proceed to prove that is indeed optimal for the offline problem. The proof is done by noting that at every point in time the algorithm runs at the minimum average speed required to complete all jobs that must be scheduled within the interval. If the machine runs at a slower speed at any time, it will not be able to finish the jobs, hence, that speed is the slowest possible speed. Furthermore, using *Earliest Deadline First* (EDF) to schedule the jobs in each interval assures that deadlines are met, and the updates of release dates and deadlines keep jobs feasible.

Using similar ideas, Yao et al. [Yao *et al.*, 1995] construct two additional algorithms

Algorithm 2.3 YAO-DEMERS-SHENKER (YDS)

```
1  set J = {J_1, ..., J_n} and T = [0, max_i{d_i}].
2  while J ≠ ∅
3      compute I* = [t_1, t_2] = arg max_{I∈T}{Δ_I}.
4      run all jobs J_i ∈ D_{I*} at speed s_i = Δ_{I*} within interval I*.
5      sort jobs in D_{I*} according to EDF.
6      for every J_i ∉ D_{I*}
7          if d_i ∈ I*
8              d_i = t_1
9          elseif r_i ∈ I*
10             r_i = t_2
11     update J = J\D_{I*}, T = T\I*.
```

for the online setting. Because in this setting we don't know which are the highest density intervals, we can only make local decisions. For every job J_i that arrives, its density $\delta_i = \frac{\rho_i}{d_i - r_i}$ is computed and at every time t the machine runs at speed $s(t) = \sum_{J_i : t \in [r_i, d_i]} \delta_i$, whereas available jobs are scheduled using EDF.

Together with the results in [Yao *et al.*, 1995], Bansal et al. [Bansal *et al.*, 2008] later showed that the competitive ratio for YDS is actually $\frac{(2-\epsilon(\alpha))^\alpha}{2}\alpha^\alpha \leq \rho \leq 2^{\alpha-1}\alpha^\alpha$, where $\epsilon(\alpha)$ is a function that converges to 0 when α goes to ∞.

Yao et al. [Yao *et al.*, 1995] also present a second online algorithm in which every time a job arrives the optimal schedule, given the current jobs, is recomputed. They don't present many results for this algorithm, but later Bansal et al. [Bansal *et al.*, 2007b] showed that this algorithm is α^α-competitive. Bansal et al. also present an additional algorithm, called BANSAL-KIMBREL-PRUHS (BKP), which uses the total available processing volume available at every time to define the speed. They prove that BKP is $2(\frac{\alpha}{\alpha-1})^\alpha e^\alpha$-competitive, which is better than the previous one for larger values of α.

Another important results by Bansal et al. [Bansal *et al.*, 2007b] is that any randomized

algorithm has a competitiveness of at least $\Omega((\frac{4}{3})^\alpha)$, implying that this problem has an inherent exponential dependence on α.

Yao et al. [Yao *et al.*, 1995] also extend their algorithm for the case in which only a discrete set of speeds $\mathbf{S} = \{\sigma_1, \ldots, \sigma_m\}$ is available. Obviously, this algorithm only works when the instance is feasible, and it works by using the YDS algorithm first and then at every point in time it alternates between two speeds using time-sharing, such that the effective final speed for that interval is equal to the one given by the YDS algorithm.

There have been other attempts to build algorithms for the case where the instance is not feasible. In that case we are interested in both reducing the energy consumption as well as minimizing the number of late jobs. The current best algorithm was presented by Bansal et al. in [Bansal *et al.*, 2010], called Slow-D. They prove that the algorithm is 4-competitive with respect to throughput and $(\alpha^\alpha + \alpha^2 4^\alpha)$-competitive with respect to energy.

2.2.2.2 Energy or QoS Budget

The other setting related to speed scaling is minimizing a scheduling performance metric with only a limited energy budget. There have not been many results in this area, and only the papers by Pruhs et al. [Pruhs *et al.*, 2007; Pruhs *et al.*, 2008] present some results. In these papers the authors study this setting considering the sum of flow times as the scheduling performance metric.

In order to tackle the problem the authors simplify the problem by considering only unit-size jobs, which decouples the scheduling problem from the speed-setting one. They also restrict themselves to the offline scenario, so all the information is available from the start. They show that in this case by using a *First Come First Serve* (FCFS) scheduling policy and solving an auxiliary LP to determine the speeds, an optimal solution can be computed in polynomial time.

Another interesting and counter-intuitive insight the authors give in [Pruhs *et al.*, 2008] is that the speed is actually not monotone functions of the energy budget. For some cases, when the energy budget is reduced, some jobs run at a faster speed in the optimal solution.

No other work has focused on this setting, thus the problem with jobs of arbitrary length remains open. The authors in [Irani and Pruhs, 2005] conjecture that using resource augmentation might help but indicate that it is fairly difficult to track the changes in speed when the energy budget is modified. The other open problem is related to the online scenario. The main difficulty in this case is to come up with an insightful definition, since the setting presented here, in an online setting has an unbounded competitiveness ratio. One can always create instances where it is impossible to know how much to spend in terms of energy at any given time so as to not violate the energy budget constraint in the future.

The reverse problem of minimizing the energy consumption having a quality of service budget has also been studied by several authors [Bansal *et al.*, 2004; Bansal *et al.*, 2007b; Bansal *et al.*, 2008; Yao *et al.*, 1995].

2.2.2.3 Balancing Energy and Scheduling Performance

The third main setting within speed scaling is the scenario in which we are interested in balancing energy costs or consumption with scheduling performance. Implicit in this case is the assumption that both energy (or any resource for that matter) and time can be (implicitly) converted into a common unit, such as dollars. There are several challenging problems to be solved in this setting and it has several direct applications, since we are able to compare directly the costs related to energy consumption and the scheduling performance, with the objective of finding a solution that achieves the best compromise between the two.

The first researchers to propose a setting in which energy and scheduling performance are balanced together were Albers and Fujiwara [Albers and Fujiwara, 2007]. Instead of having an energy budget they consider minimizing the total energy consumption plus the total flow time, i.e. $\sum_i E(s_i) + F_i$, with nonpreemptive schedules. The authors first simplify the problem by assuming that all jobs are unit sized and proceed to describe a *Dynamic Programming* (DP) based algorithm that finds the optimal solution for the offline scenario.

Another interesting result by Albers and Fujiwara in [Albers and Fujiwara, 2007] is that they show that any deterministic online algorithm has a competitive ratio of at least $\Omega(n^{1-\frac{1}{\alpha}})$.

Considering that for the standard total flow time scheduling problem no online algorithm can perform better than $\Omega(n)$-competitive [Albers and Fujiwara, 2007], this result indicates that the additional flexibility given by the speeds does not help significantly to correct bad scheduling decisions in the online setting.

Albers and Fujiwara also present in [Albers and Fujiwara, 2007] a constant competitive ratio algorithm for the case with unit sized jobs. The algorithm uses different phases in which it processes jobs in batches. At any given phase the algorithm schedules optimally the jobs of the current batch, while it collects the new arriving jobs. The speed at which jobs in batch i run at any given time is set to $\sqrt[\alpha]{\frac{n_i}{c}}$, where n_i is the number of unfinished jobs in the batch and $c = \alpha - 1$ when $\alpha < \frac{19+\sqrt{161}}{10}$ or $c = 1$ otherwise. Once the jobs of the current batch are finished the algorithm sorts the collected jobs and repeats the procedure. The authors show that this algorithm, which they call PHASEBAL is $8.3e(1 + \Phi)^\alpha$-competitive, where $\Phi = \frac{1+\sqrt{5}}{2}$.

Later, Bansal et al. [Bansal *et al.*, 2009] were able to improve this result, but considering the preemptive version. At any point in time they schedule jobs according to the *Shortest Remaining Processing Time* (SRPT) rule and run the job at speed $\sqrt[\alpha]{n_t + 1}$ where n_t is the number of active jobs at time t, that is the number of jobs that have arrived but not yet finished. Through the use of potential functions they are able to show that this algorithm is $(3 + \epsilon)$-competitive even when jobs have arbitrary lengths and the power function is regular. They define regular power functions $P(s)$ as those that are: continuous and differentiable in $[0, \infty)$, $P(0) = 0$, strictly increasing, strictly convex, and unbounded.

The current best result is by Andrew et al. [Andrew *et al.*, 2009]. They also use SRPT as the scheduling policy but the speed at which they run the jobs is defined by $\sqrt[\alpha]{n_t}$. They show that this algorithm is $(2 + \epsilon)$-competitive and they only require that the power function is non-negative and unbounded. Furthermore, they show that this result is tight.

Andrew et al. showed in [Andrew *et al.*, 2010] that not only SRPT works well, but also *Processor Sharing* (PS) and even a gated-static speed scaling performs close to what is achieved by SRPT in practice. The key insight in their paper is that although gated-static speed scaling performs well, dynamic speed scaling (using $\sqrt[\alpha]{n_t}$) provides robustness when

Problem	algorithm	approx. ratio	bound
$1\|r_i, \rho_i = 1\| \sum E(s_i) + F_i$	DP	1	-
$1\|r_i\| \sum E(s_i) + F_i$	?	?	?
$1\|prec\| \sum E_i(s_i) + w_i C_i$	SAIAS	$4(1+\epsilon)(1+\delta)$	?
$1\|r_i, prec\| \sum E_i(s_i) + w_i C_i$	SAIAS	$(3+2\sqrt{2})(1+\epsilon)(1+\delta)$	?
$1\|prec\| \sum E_i(s_i) + w_i T_i$	SAIAS-T	$4^{\alpha}(1+\epsilon)^{\alpha-1}(1+\delta)^{\alpha-1}$	?

Table 2.1: Summary of known results - Offline Setting

workloads are uncertain.

It is important to note that all current research directions for this setting are based on some simple order policy for the schedule (like SRPT) and a separate policy for the speed at which to run each job.

Tables 2.1 and 2.2 show a summary of the current best results for several different problems, in the offline and online settings respectively.

The work I present in this thesis is focused mostly in the energy balancing setting discussed here, although we generalize it further by allowing very general and job-dependent energy cost functions. Although having job-dependent energy cost functions does not provide advantages in the CPU energy aware setting, since all jobs are processed in the same CPU and thus have the same energy consumption rate, this will be the key for us to extend the use of our algorithms to the more general setting discussed in Chapter 5.

The prior work on speed scaling algorithms also assumes that the energy cost is *only*

Problem	algorithm	competit. ratio	bound
$1\|r_i, \rho_i = 1\| \sum E(s_i) + F_i$	PHASEBAL	$8.3e(1+\Phi)^{\alpha}$	?
$1\|r_i\| \sum E(s_i) + F_i$	?	?	$\Omega(n^{1-\frac{1}{\alpha}})$
$1\|r_i, pmtn\| \sum E(s_i) + F_i$	$(SRPT, P^{-1}(n))$	$2+\epsilon$	2

Table 2.2: Summary of known results - Online Setting

a function of the speed. We allow for the cost to be dependent on *all* the resources being utilized. For example, in the context of scheduling computational task, we can allow for the cost to be dependent on the CPU speed, the RAM utilized, and bus bandwidth and speed.

This page is intentionally printed only with this statement.

Chapter 3

Energy Aware Scheduling

The basic problem that we will explore and extend throughout this thesis is the energy aware scheduling (EAS). In this setting only the energy consumed by processing each job is taken into account, and the energy consumption rate will determine the speed at which the jobs are processed. In this chapter we will discuss this basic setting and develop algorithms to compute approximate optimal solutions.

In Section 3.1 we will first assume that the energy cost is of the polynomial form generally used in the EAS literature, and in this particular setting we will present two approximation algorithms for the case when the total weighted completion time is the scheduling performance metric.

Next, in Section 3.2 we will modify one of the algorithms described in Section 3.1 to include another important scheduling metric: total weighted tardiness. This will require important changes that will later allow us to build the general scheduling algorithm presented in Chapter 7.

Finally, in Section 3.3 we will generalize all the results for a larger class of energy cost functions, which require only minor regularity conditions.

For convenience we will use an extended version of the notation of Graham et al. [Graham *et al.*, 1979] to refer to our different resource cost aware scheduling problems, i.e. $1|r_i, prec|\sum \mathcal{R}_i(\boldsymbol{\psi}^{(i)}) + w_i C_i$, will refer to the problem setting with 1 machine, with r_i re-

lease dates, precedence constraints, and the weighted completion time as the scheduling performance metric. The $\mathcal{R}_i(\boldsymbol{\psi}^{(i)})$ term indicates that the resource cost is also added to the performance metric, whereas a term $E_i(\boldsymbol{\psi}^{(i)})$ or $\mathcal{E}_i(\boldsymbol{\psi}^{(i)})$ indicates that only energy cost is used. These terms will be further explained in the following sections.

3.1 Total Weighted Completion Time

In this section we consider the EAS problem with the commonly studied scheduling metric, *weighted completion time*. This metric has not received attention in the resource or energy cost aware scheduling literature, even though it has applications in several different areas such as software compilers, instruction scheduling in VLIW processors, MapReduce-like systems, manufacturing processes, and maintenance procedures among others [Chang *et al.*, 2011; Chekuri and Khanna, 2004; Chekuri *et al.*, 2001; Pinedo, 2008]. In all these applications there are related resources that can be used to control the speed at which jobs are processed and need to be taken into account. For example in maintenance and repair procedures, the processing time of a job can depend on the workforce size, spare parts inventory levels, energy consumption, training, etc. Furthermore, in many settings jobs have precedence constraints as well, something that has not been dealt with in the current literature.

Given a schedule in which job i with weight w_i and release time r_i is completed at time C_i, the total weighted completion time is given by $\sum_i w_i C_i$. We consider the non-preemptive, offline problem on one machine, and allow arbitrary precedence constraints and arbitrary release dates as well. Our objective is to minimize the sum of our scheduling metric and the total energy consumption cost. We are not aware of any previous work on energy aware scheduling algorithms for this metric, although there is a rich literature on minimizing weighted completion time in the absence of energy concerns (e.g. [Phillips *et al.*, 1998; Pinedo, 2008; Skutella, 2006]).

Minimizing weighted completion time is well studied in the combinatorial scheduling literature. Phillips et al. [Phillips *et al.*, 1998] and Hall et al. [Hall *et al.*, 1997; Hall *et al.*,

1996] introduced the concept of α-*points* that has lead to small constant factor approximation algorithms for many scheduling problems [Skutella, 2006]. In the α-point approach, the scheduling problem is formulated as an integer program in terms of decision variables x_{it} that is 1 if job i completes at time t. The α-point of each job is defined as the earliest time at which an α fraction of the job has completed in the linear relaxation. The jobs are ordered in the order of their α-points and run in non-preemptive fashion. There are many variants and extensions of these technique including choosing α randomly [Chekuri *et al.*, 2001; Goemans, 1997] or choosing a different α for each job [Goemans *et al.*, 2002]. We extend α-point technique to the speed-scaling setting by defining α-*speeds*, which are achieved by time-sharing between the available speeds.

Our approach allows job-dependent power functions, and thus can be applied to a more general class of problems outside this specific setting. Furthermore, most energy aware algorithms assume cost functions that are closely related to energy consumption; however in practice, the actual energy cost is not simply a function of energy consumption, it is a complicated function of discounts, pricing, time of consumption, peak energy requirements, etc. That observation motivated our consideration of a more general class of cost functions that are only restricted to be non-negative. We are not aware of any other work that allows such general costs.

3.1.1 Problem Formulation

The problem setting is as follows. We are given n jobs, where job i, $i = \{1, \ldots, n\}$, has a processing requirement of $\rho_i \in \mathbb{N}_+$ machine cycles, a release time r_i, and an associated positive weight w_j. Let s_i denote the speed at which job i runs on the machine and let C_i denote its completion time. Let $\Pi = \{\pi(1), \ldots, \pi(n)\}$ denote the order in which the jobs are processed, i.e. $\pi(i) = k$ implies that job k is the i-th job to be processed. Then the completion time of the i-th job to be processed is $C_{\pi(i)} = \max\{r_{\pi(i)}, C_{\pi(i-1)}\} + \frac{\rho_{\pi(i)}}{s_{\pi(i)}}$, with $\pi(0) = 0$ and $C_{\pi(0)} = 0$. We assume that preemption is not allowed.

Let $E_i(s_i)$ denote the energy cost of running job i at speed s_i. Initially we consider

$E_i(s_i) = v_i \rho_i s_i^{\beta-1}$, where $\beta \geq 2$ and v_i are known constants. Later we show that our algorithms work for more general energy cost functions $\mathcal{E}_i(s_i)$ that only require minor regularity conditions.

The objective is to compute a feasible schedule, consisting of an order Π that respects the possible precedence constraints and/or release date constraints, and the vector of job speeds $\mathbf{s} = \{s_1, \ldots, s_n\} \in \mathbb{R}_+^n$ that minimizes the total cost,

$$f(\Pi, \mathbf{s}) = \sum_{i=1}^{n} \left[v_{\pi(i)} \rho_{\pi(i)} s_{\pi(i)}^{\beta-1} + w_{\pi(i)} C_{\pi(i)} \right]. \tag{3.1}$$

Note that the energy consumed by job $\pi(i)$ only depends on the speed $s_{\pi(i)}$, whereas its completion time depends on the speeds of all jobs $\pi(j)$ with $j < i$. Since the cost function is convex we can assume, without loss of generality, that each job runs at a constant speed. Furthermore, since we assume that preemption is not allowed, once a job is selected for processing, it is not interrupted to process another job.

In order to use an indexed-based formulation for this problem, we will first compute the set of speeds in which the optimal speeds are, which we will later quantize.

3.1.2 Speed Bounds

In order to quantize the set of speeds for our algorithms, it is necessary to find a set that contains the optimal speed solutions, otherwise the final solution computed by the algorithm can be arbitrarily away from the optimal. The following lemma bounds the set of possible optimal speeds, and thus determines the set we need to quantize.

Lemma 3.1. *The optimal speed s_i^* for any job i belongs to the interval $[\sigma_{\min}, \sigma_{\max}]$, where $\sigma_{\min} = \min_j \left\{ \sqrt[\beta]{\frac{w_j}{(\beta-1)v_j}} \right\} > 0$ and $\sigma_{\max} = \max_j \left\{ \sqrt[\beta]{\frac{\sum_{k=1}^{n} w_k}{(\beta-1)v_j}} \right\} < \infty$.*

Proof. First fix an arbitrary schedule $\Pi = \{\pi(1), \pi(2), \ldots, \pi(n)\}$ that satisfies all precedence constraints. The following optimization problem computes the optimal speeds given that

schedule:

$$\min_{\mathbf{s}} \sum_{i=1}^{n} v_{\pi(i)} \rho_{\pi(i)} s_{\pi(i)}^{\beta-1} + w_{\pi(i)} C_{\pi(i)} \tag{3.2}$$

s.t.:

$$\begin{aligned} C_{\pi(i)} &\geq r_{\pi(j)} + \textstyle\sum_{k=j}^{i} \frac{\rho_{\pi(k)}}{s_{\pi(k)}}, \quad \forall j \in \{1,\dots,i\},\ \forall i \in \{1,\dots,n\} \\ s_{\pi(i)} &\geq 0, \qquad\qquad\qquad\qquad \forall i \in \{1,\dots,n\}. \end{aligned}$$

Let $\lambda_{\pi(i)\pi(j)}$, $j = \{1,\dots,i\}$ define the dual variables of the release date constraints in (3.2) for the i-th job, $i = \{1,\dots,n\}$. From the necessary optimality conditions we get that

$$\sum_{j=1}^{i} \lambda^*_{\pi(i)\pi(j)} = w_{\pi(i)}, \ \forall i \in \{1,\dots,n\}, \tag{3.3}$$

and thus,

$$s^*_{\pi(i)} = \sqrt[\beta]{\frac{\sum_{j=i}^{n} \sum_{k=1}^{i} \lambda^*_{\pi(j)\pi(k)}}{(\beta-1) v_{\pi(i)}}}, \ \forall i \in \{1,\dots,n\}, \tag{3.4}$$

where $\mathbf{s}^*$ and λ^* are the optimal speeds and optimal dual variables respectively. Note from (3.4) that the optimal speed of the i-th job only depends of the dual variables of the completion time constraints of future jobs, and not past ones.

If there is idle time between the i-th and $i+1$-st job, then $C_{\pi(j)} > r_{\pi(k)} + \sum_{l=j}^{i} \frac{\rho_{\pi(l)}}{s_{\pi(l)}}$, for $j = \{i+1,\dots,n\}$ and $k = \{1,\dots,i\}$. Thus, by the complementary slackness conditions we get that $\lambda_{\pi(j)\pi(k)} = 0$, for $j = \{i+1,\dots,n\}$ and $k = \{1,\dots,i\}$. Then, from (3.4), the smallest optimal speed that the i-th job can have is attained when there is idle time after it, and thus

$$s^*_{\pi(i)} = \sqrt[\beta]{\frac{w_{\pi(i)}}{(\beta-1) v_{\pi(i)}}},$$

which is positive since $w_{\pi(i)} > 0$ and $v_{\pi(i)}$ is finite. By setting $\sigma_{\min} = \min_j \left\{ \sqrt[\beta]{\frac{w_j}{(\beta-1)v_j}} \right\}$, we obtain that $s^*_i \geq \sigma_{\min} > 0$, $\forall i$, and for any order Π.

On the other hand, from (3.3) we get that $\sum_{k=1}^{i} \lambda^*_{\pi(j)\pi(k)} \leq w_{\pi(j)}$ for $j \geq i$. Hence from (3.4) we get that

$$s^*_{\pi(i)} \leq \sqrt[\beta]{\frac{\sum_{j=1}^{n} w_j}{(\beta-1) v_{\pi(i)}}} < \infty,$$

since w_j is finite $\forall j$ and $v_{\pi(i)} > 0$. By setting $\sigma_{\max} = \max_j \left\{ \sqrt[\beta]{\frac{\sum_{i=1}^{n} w_i}{(\beta-1)v_j}} \right\}$ we have $s^*_i \leq \sigma_{\max} < \infty$, $\forall i$ and any order Π. □

It is important to notice that both bounds are tight. As shown in the previous proof, the lower bound is attained when the job with the smallest $\frac{w_j}{v_j}$ has idle time after it. On the other hand, the upper bound is attained when the job with the smallest v_i is scheduled first, and the release dates are small enough such that $r_{\pi(i)} < \sum_{j=1}^{i} \frac{\rho_{\pi(j)}}{s_{\pi(j)}}$, which also implies that there is no idle time between jobs. In this case the first job will run at speed $\sigma_{\max}$.

Corollary 3.1. *The maximum speed $\sigma_{\max}$ is bounded by*

$$\sigma_{\max} \leq \sigma_{\min} \sqrt[\beta]{\frac{n w_{\max} v_{\max}}{w_{\min} v_{\min}}},$$

where $w_{\max} = \max_i\{w_i\}$, $w_{\min} = \min_i\{w_i\}$, $v_{\max} = \max_i\{v_i\}$ and $v_{\min} = \min_i\{v_i\}$.

Proof. From Lemma 3.1 it follows,

$$\sigma_{\min} = \min_j \left\{ \sqrt[\beta]{\frac{w_j}{(\beta-1)v_j}} \right\} \geq \sqrt[\beta]{\frac{w_{\min}}{(\beta-1)v_{\max}}}.$$

Similarly, the maximum speed is bounded by

$$\sigma_{\max} = \max_j \left\{ \sqrt[\beta]{\frac{\sum_{i=1}^{n} w_i}{(\beta-1)v_j}} \right\} \leq \sqrt[\beta]{\frac{n w_{\max}}{(\beta-1)v_{\min}}}.$$

Therefore, it follows that,

$$\frac{\sigma_{\max}}{\sigma_{\min}} \leq \sqrt[\beta]{\frac{n w_{\max} v_{\max}}{w_{\min} v_{\min}}}.$$

□

3.1.3 Speed Quantization

We now quantize the set of feasible speeds $[\sigma_{\min}, \sigma_{\max}]$ into m possible speeds $\mathbf{S} = \{\sigma_1, \ldots, \sigma_m\}$. Using Lemma 3.1 we determine the maximum and minimum speeds at which any job can be processed, $\sigma_{\max}$ and $\sigma_{\min}$, and we can define speed $\sigma_j = \frac{1}{k_j}\sigma_{\max}$, where $k_j \in \mathbb{N}^+$, for $j = 1, \ldots, m$. The restriction that $k_j \in \mathbb{N}^+$ is required for the time-indexed formulation described in Section 3.1.5, but it will be relaxed to $k_j \in \mathbb{R}^+$ for the interval-indexed formulation presented in Section 3.1.6. W.l.o.g. we assume that $k_i > k_j$, $\forall i < j$, and thus the speeds σ_j

are ordered in increasing values. To ensure that the range of speeds given by Lemma 3.1 is covered, we set $k_m = 1$, so $\sigma_m = \sigma_{\max}$ and set $k_1 \geq \frac{\sigma_{\max}}{\sigma_{\min}}$ so $\sigma_1 \leq \sigma_{\min}$.

Given that the quantization of the feasible speed set induces an error for all our formulations when compared to the continuous speed case, we first bound this error as follows. Let the quantization be in geometrically increasing speed values, i.e. $k_j = (1+\delta)^{m-j}$ for some $\delta > 0$, and thus $\sigma_{j+1} = (1+\delta)\sigma_j$. Note that given δ we define m such that,

$$m = \left\lceil \frac{\log(\sigma_{\max}) - \log(\sigma_{\min})}{\log(1+\delta)} \right\rceil + 1, \tag{3.5}$$

to make sure that the whole feasible speed set from Lemma 3.1 is covered.

The following Lemma bounds the error induced by the quantization assuming that $k_j \in \mathbb{R}^+$, but a similar argument can be used when $k_j \in \mathbb{N}^+$.

Lemma 3.2. *The optimal value of (3.1) using the previous speed quantization scheme, i.e.* $\mathbf{S} = \{\sigma_1, \ldots, \sigma_m\}$*, with* $\sigma_j = \frac{\sigma_{\max}}{(1+\delta)^{m-j}}$*, and* m *as in (3.5), is at most* $(1+\frac{\delta}{2})^{\beta-1}$ *times the optimal value of (3.1) with continuous speeds.*

Proof. Let Π^* and $\mathbf{s}^*$ denote an optimal solution of (3.1) without speed quantization. W.l.o.g. we assume that $\Pi^* = \{1, \ldots, n\}$. Let $\tilde{\mathbf{s}}$ denote the element-wise rounding of the solution $\mathbf{s}^*$ to the nearest speed $\sigma_j \in \mathbf{S}$. Since precedence and release date constraints are not violated by the rounding procedure, $\tilde{\mathbf{s}}$ is still feasible.

First assume that all speeds are rounded up. In this case, if $s_i^* \in [\sigma_j, \sigma_{j+1}]$, then it must be that $s_i^* \geq \frac{1}{2}(\sigma_j + \sigma_{j+1}) = \sigma_j(1+\frac{\delta}{2})$ for s_i^* to be rounded up. Hence, the rounding procedure sets $\tilde{\mathbf{s}}_i$ at most

$$\tilde{s}_i \leq \left(1 + \frac{\delta}{2+\delta}\right) s_i^*.$$

Since speeds increase, completion times do not increase. On the other hand energy consumption increases, but it is bounded as follows,

$$E_i(\tilde{s}_i) \leq \left(1 + \frac{\delta}{2+\delta}\right)^{\beta-1} E_i(s_i^*) \;\Rightarrow\; \sum_{i=1}^{n} E_i(\tilde{s}_i) \leq \left(1 + \frac{\delta}{2+\delta}\right)^{\beta-1} \sum_{i=1}^{n} E_i(s_i^*), \tag{3.6}$$

thus, for this case, $f(\Pi^*, \tilde{\mathbf{s}}) \leq \left(1 + \frac{\delta}{2+\delta}\right)^{\beta-1} f(\Pi^*, \mathbf{s}^*)$.

Next, we assume that all speeds are rounded down. Then if $s_i^* \in [\sigma_j, \sigma_{j+1}]$, it must be that $s_i^* \leq \sigma_j(1+\frac{\delta}{2})$, and the rounding procedure reduces each speed at most,

$$\tilde{s}_i \geq \left(\frac{2}{2+\delta}\right) s_i^*.$$

Let C^* define the optimal completion times when no quantization is used, and $\tilde{C}$ when speeds are quantized. Because speed decreases the energy cost does not worsen, whereas the completion times will increase, but can be bounded as follows,

$$\tilde{C}_i = \max\{r_i, \tilde{C}_{i-1}\} + \frac{\rho_i}{\tilde{s}_i} \leq \left(1+\frac{\delta}{2}\right)\left(\max\{r_i, C^*_{i-1}\} + \frac{\rho_i}{s_i^*}\right) = \left(1+\frac{\delta}{2}\right) C_i^* \tag{3.7}$$

$$\Rightarrow \quad \sum_{i=1}^{n} w_i \tilde{C}_i \leq \left(1+\frac{\delta}{2}\right) \sum_{i=1}^{n} w_i C_i^*, \tag{3.8}$$

where the last inequality follows by recursion on the job index. Hence, for this case, $f(\Pi^*, \tilde{\mathbf{s}}) \leq \left(1+\frac{\delta}{2}\right) f(\Pi^*, \mathbf{s}^*)$.

From (3.6) and (3.8) it follows that for the general case,

$$f(\Pi^*, \tilde{\mathbf{s}}) \leq \max\left\{\left(1+\frac{\delta}{2+\delta}\right)^{\beta-1}, \left(1+\frac{\delta}{2}\right)\right\} f(\Pi^*, \mathbf{s}^*) \leq \left(1+\frac{\delta}{2}\right)^{\beta-1} f(\Pi^*, \mathbf{s}^*). \tag{3.9}$$

Since $\min_{\mathbf{s}\in\mathbf{S}} f(\Pi^*, \mathbf{s}) \leq f(\Pi^*, \tilde{\mathbf{s}})$, the lemma follows. □

It is important to note that in many contexts the feasible set of speeds is already quantized beforehand, e.g. CPUs generally allow only a discrete set of speeds. In such situations the quantization will add no error to the solution at this stage. However, as we will present in the subsequent sections, the output of our algorithms not necessarily returns speeds that belong to the discrete set $\mathbf{S}$. Thus, we need to round the final solution to the given speed set, and the previous approximation error will reappear.

3.1.4 Cases Solvable in Polynomial Time

When no precedence constraints and release dates exist, there are two versions of this problem that can be optimally solved in polynomial time: when all weights w_i are equal, and when all

jobs are of the same size (i.e. $\rho_i = \rho,\ \forall i$) and have the same energy cost function. For these cases we have the following result:

Theorem 3.1. *If* $w_i = w,\ \forall i$ *or* $\rho_i v_i^{\frac{1}{\beta}} = \xi,\ \forall i$ *then the order* Π *is optimal if*

$$\frac{w_{\pi(i)}}{\rho_{\pi(i)} v_{\pi(i)}^{\frac{1}{\beta}}} \geq \frac{w_{\pi(i+1)}}{\rho_{\pi(i+1)} v_{\pi(i+1)}^{\frac{1}{\beta}}},\ \forall i \in \{1, \ldots, n-1\}.$$

Proof. Define $\xi_i \equiv \rho_i v_i^{\frac{1}{\beta}}$. Using the dual formulation of (3.2) with no precedence or release date constraints, it follows that the optimization problem is given by

$$\min_{\pi} G(\pi) = \min_{\pi} \left\{ \sum_{i=1}^{n} \mathcal{B} \xi_{\pi(i)} \left(\sum_{j=i}^{n} w_{\pi(j)} \right)^b \right\}, \tag{3.10}$$

where $\mathcal{B} \equiv \frac{\beta}{(\beta-1)^b}$ and $b \equiv \frac{\beta-1}{\beta}$.

First, when $w_i \equiv w$, Theorem 3.1 implies that in the optimal order $\xi_{\pi(i+1)} \geq \xi_{\pi(i)}$. By contradiction, let π be an optimal order such that for some index k, $\xi_{\pi(k+1)} < \xi_{\pi(k)}$. For this order the total cost is

$$\begin{aligned} G(\pi) &= \sum_{i=1}^{n} \mathcal{B} \xi_{\pi(i)} \left(\sum_{j=i}^{n} w \right)^b = \sum_{i=1}^{n} \mathcal{B} \xi_{\pi(i)} \left((n-i+1) w \right)^b \\ &= \mathcal{B} w^b \left\{ \xi_{\pi(k)} (n-k+1)^b + \xi_{\pi(k+1)} (n-k)^b + \sum_{\substack{i=1 \\ i \neq k, k+1}}^{n} \xi_{\pi(i)} (n-i+1)^b \right\}. \end{aligned}$$

Let π_k define the order where we switch jobs k and $k+1$ from order π, i.e. $\pi_k(k) = \pi(k+1)$ and $\pi_k(k+1) = \pi(k)$. Given this order we have that

$$\begin{aligned} G(\pi) - G(\pi_k) &= \mathcal{B} w^b \left\{ \xi_{\pi(k)} (n-k+1)^b + \xi_{\pi(k+1)} (n-k)^b - \xi_{\pi(k+1)} (n-k+1)^b \right. \\ &\quad \left. - \xi_{\pi(k)} (n-k)^b \right\} \\ &= \mathcal{B} w^b \left\{ (n-k+1)^b (\xi_{\pi(k)} - \xi_{\pi(k+1)}) - (n-k)^b (\xi_{\pi(k)} - \xi_{\pi(k+1)}) \right\} \\ &= \mathcal{B} w^b \left\{ \left(\xi_{\pi(k)} - \xi_{\pi(k+1)} \right) \left((n-k+1)^b - (n-k)^b \right) \right\}. \end{aligned}$$

By our initial assumption, the first term is positive (since $\xi_{\pi(k+1)} < \xi_{\pi(k)}$) and the second one is always positive, hence $G(\pi) - G(\pi_k) > 0$ which is a contradiction, since that implies that π_k has a smaller cost.

When $\xi_i \equiv \xi$, Theorem 3.1 implies that an order π is optimal then $w_{\pi(i)} \geq w_{\pi(i+1)}$. Let π be an optimal order such that for some index k, $w_{\pi(k)} < w_{\pi(k+1)}$. The total cost for this solution is

$$\begin{aligned} G(\pi) &= \sum_{i=1}^{n} \mathcal{B}\xi \left(\sum_{j=i}^{n} w_{\pi(i)} \right)^b \\ &= \mathcal{B}\xi \left\{ \sum_{i=1}^{k} \left(\sum_{j=i}^{n} w_{\pi(i)} \right)^b + \left(\sum_{j=k+1}^{n} w_{\pi(i)} \right)^b + \sum_{i=k+2}^{n} \left(\sum_{j=i}^{n} w_{\pi(i)} \right)^b \right\} \\ &= \mathcal{B}\xi \left\{ \sum_{i=1}^{k} \left(\sum_{j=i}^{n} w_{\pi(i)} \right)^b + \left(w_{\pi(k+1)} + \sum_{j=k+2}^{n} w_{\pi(i)} \right)^b + \sum_{i=k+2}^{n} \left(\sum_{j=i}^{n} w_{\pi(i)} \right)^b \right\}. \end{aligned}$$

Let π_k define the order where we switch jobs k and $k+1$ from order π. Given this new order we have

$$G(\pi) - G(\pi_k) = \mathcal{B}\xi \left\{ \left(w_{\pi(k+1)} + \sum_{j=k+2}^{n} w_{\pi(i)} \right)^b - \left(w_{\pi(k)} + \sum_{j=k+2}^{n} w_{\pi(i)} \right)^b \right\} > 0,$$

since $w_{\pi(k+1)} > w_{\pi(k)}$ by our initial assumption, which is a contradiction since this result implies that order π_k has a lower cost. □

3.1.5 Time-and-Speed-Indexed Formulation

We now present a time-and-speed-indexed formulation, which is an extension of the integer formulation presented in [Sousa and Wolsey, 1992]. Then, we extend the algorithms and results obtained in [Hall *et al.*, 1996] to the energy aware setting.

Given that this formulation requires time to be divided into constant size pieces and index all variables by these time steps, the resulting algorithm might not be polynomial en the size of the input. Although this is a major issue, the problem will be corrected with the formulation presented in Section 3.1.6.

3.1.5.1 Model Description and Approximation Algorithm

In this first formulation we discretize time in steps of size $\tau = \frac{1}{\sigma_{\max}}$, which is the minimum size time-step a job can have, and use a time-step-and-speed-indexed formulation for problem (3.1).

Let $p_{ij} = \frac{\rho_i}{\sigma_j}$ denote the processing time of job i at speed σ_j. Then job i requires $\hat{p}_{ij} = \frac{p_{ij}}{\tau} = k_j \rho_i$ time steps, each of size τ. Since $k_j \in \mathbb{N}^+$ and $\rho_i \in \mathbb{N}^+$, then $\hat{p}_{ij} \in \mathbb{N}^+$.

For a given instance of (3.1), let T define an upper bound on the total number of time steps of the schedule, e.g. we can set $T = \frac{1}{\tau} \max_{i=1}^{n} r_i + \sum_{i=1}^{n} \frac{p_{i1}}{\tau}$.

Let

$$y_{ijt} = \begin{cases} 1, & \text{if job } i \text{ runs at a speed } \sigma_j \text{ and completes in time step } t \\ 0, & \text{otherwise.} \end{cases} \tag{3.11}$$

Using the y_{ijt} variables the objective function in (3.1) can be rewritten as:

$$\min_{\mathbf{y}} \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{t=1}^{T} \left(v_i \rho_i \sigma_j^{\beta-1} + w_i \tau t \right) y_{ijt}. \tag{3.12}$$

Next we list the constraints on the decision variables $\{y_{ijt}\}$:

1. Each job runs at a unique speed and is completed at a unique time step, i.e. for all $i = \{1, \ldots, n\}$:

$$\sum_{j=1}^{m} \sum_{t=1}^{T} y_{ijt} = 1. \tag{3.13}$$

2. A job i, $i = \{1, \ldots, n\}$ running at speed σ_j, $j = \{1, \ldots, m\}$, is in process at time step t if it is completed within the interval $[t, t + \hat{p}_{ij} - 1]$. Since the machine can only process one job on each time step we must have that for all $t = \{1, \ldots, T\}$:

$$\sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{u=t}^{\min\{T, t+\hat{p}_{ij}-1\}} y_{iju} \leq 1. \tag{3.14}$$

3. y_{ijt} must be a binary variable for all $i = \{1, \ldots, n\}$, $j = \{1, \ldots, m\}$, and $t = \{1, \ldots, T\}$,

$$y_{ijt} \in \{0, 1\}. \tag{3.15}$$

4. Job i running at speed σ_j requires $\hat{p}_{ij} = k_j\rho_i$ time steps to be processed after its release time-step $\frac{r_i}{\tau}$, thus for all $i = \{1, \ldots, n\}$ and $j = \{1, \ldots, m\}$:

$$y_{ijt} = 0, \quad \text{if } t < \frac{r_i}{\tau} + k_j\rho_i. \tag{3.16}$$

5. For the precedence constraints, first note that $\sum_{j=1}^{m}\sum_{s=1}^{t} y_{ijs} = 1$ if and only if job i has been completed by time step t. Now $i_1 \prec i_2$ means that if job i_2 runs at speed σ_j, then job i_1 must be finished at least $\hat{p}_{i_2j}$ time steps before job i_2, thus the precedence constraint $i_1 \prec i_2$ can be written as, for $t = 1, \ldots, T - \hat{p}_{i_2m}$:

$$\sum_{j=1}^{m}\sum_{u=1}^{t} y_{i_1ju} \geq \sum_{j=1}^{m}\sum_{u=1}^{\min\{t+\hat{p}_{i_2j},T\}} y_{i_2ju}. \tag{3.17}$$

We now describe a new approximation algorithm for the EAS problem, called SCHEDULE BY α-POINTS AND α-SPEEDS (SAPAS), which is detailed in Algorithm 3.1.

Let $\bar{y}_{ijt}$ denote the optimal solution of the linear relaxation of (3.12)-(3.17), where the binary constraint $y_{ijt} \in \{0, 1\}$ is relaxed to $y_{ijt} \geq 0$. In step 1 of the algorithm we compute the optimal solution $\bar{\mathbf{y}}$ and then, in step 2, for any given $0 \leq \alpha \leq 1$ we define the α-point of job i as

$$t_i^{\alpha} = \min\left\{t : \sum_{j=1}^{m}\sum_{u=1}^{t} \bar{y}_{iju} \geq \alpha\right\}, \tag{3.18}$$

which is the earliest time step after which an α-fraction of job i has been completed.

Next, in step 3, we compute the α-speeds as follows. Since $\sum_{j=1}^{m}\sum_{u=1}^{t_i^{\alpha}} \bar{y}_{iju} \geq \alpha$, we define the auxiliary variable $\{\tilde{y}_{ijt}\}_{i=1,j=1,t=1}^{n,n,T}$ as follows, to ensure that $\sum_{j=1}^{m}\sum_{u=1}^{t_i^{\alpha}} \tilde{y}_{iju} = \alpha$:

$$\tilde{y}_{ijt} = \begin{cases} \bar{y}_{ijt} & , t < t_i^{\alpha} \\ \max\left\{\min\left\{\bar{y}_{ijt_i^{\alpha}}, \alpha - \sum_{l=1}^{j-1} \bar{y}_{ilt_i^{\alpha}} - \beta_i\right\}, 0\right\} & , t = t_i^{\alpha} \\ 0 & , t > t_i^{\alpha}, \end{cases} \tag{3.19}$$

where $\beta_i = \sum_{j=1}^{m}\sum_{u=1}^{t_i^{\alpha}-1} \bar{y}_{iju} < \alpha$. This new auxiliary variable preserves the values of $\bar{y}_{ijt}$ for all time steps $t < t_i^{\alpha}$, and for time step $t = t_i^{\alpha}$ it only considers the values of $\bar{y}_{ijt}$ that sum up to α, starting from the slowest speed first, i.e. $j = 1$.

Algorithm 3.1 SCHEDULE BY α-POINTS AND α-SPEEDS (SAPAS) FOR EAS

Inputs: set of jobs, $\alpha \in (0,1)$, τ, set of speeds $\mathbf{S} = \{\sigma_1, \ldots, \sigma_m\}$.

1 Compute an optimal solution $\bar{\mathbf{y}}$ to the linear relaxation (3.12)-(3.17).

2 Compute the α-points $\mathbf{t}^\alpha$ and compute an order Π^α that has the jobs ordered in non-decreasing values of t_i^α.

3 Compute the α-speeds $\mathbf{s}^\alpha$.

4 Round down each s_i^α to the nearest speed in $\mathbf{S}$ and run job i at this rounded speed, $\bar{s}_i^\alpha$.

5 Set the i-th job to start at time $\max\{r_{\pi(i)}, \bar{C}^\alpha_{\pi(i-1)}\}$, where $\bar{C}^\alpha_{\pi(i-1)}$ is the completion time of the previous job using the rounded α-speeds, and $\bar{C}^\alpha_{\pi(0)} = 0$.

6 **return** speeds $\bar{\mathbf{s}}^\alpha$, order Π^α, and completion times $\bar{\mathbf{C}}^\alpha$.

We now define a probability mass function (pmf) $\mu_i = (\mu_{i1}, \ldots, \mu_{im})$ on the set of speeds $\mathbf{S} = \{\sigma_1, \ldots, \sigma_m\}$ as follows,

$$\mu_{ij} = \frac{1}{\alpha} \sum_{u=1}^{t_i^\alpha} \tilde{y}_{iju}. \tag{3.20}$$

Let $\hat{s}_i$ define a random variable distributed according to the pmf μ_i. Then, the α-speed of job i, s_i^α, is defined as follows,

$$\frac{1}{s_i^\alpha} = \mathbb{E}\left[\frac{1}{\hat{s}_i}\right] = \sum_{j=1}^{m} \frac{\mu_{ij}}{\sigma_j} \Rightarrow s_i^\alpha = \frac{1}{\mathbb{E}\left[\frac{1}{\hat{s}_i}\right]}, \tag{3.21}$$

where the definition is done over the reciprocal of the speeds since the completion times are proportional to these reciprocals instead of the speeds, and this proportionality is required in the analysis of the algorithm.

In step 4 of the algorithm we round down the α-speeds since s_i^α might not belong to the set of possible speeds. The following lemma bounds the error introduced by this rounding.

Lemma 3.3. *The cost of the solution with the rounded down speeds $\bar{\mathbf{s}}^\alpha$ is at most $(1+\delta)$ times the cost of the solution using the α-speeds $\mathbf{s}^\alpha$.*

Proof. The energy cost function $E_i(s_i)$ is increasing so rounding down does not increase the energy cost, but the completion time is now larger. Let C_i^α be the completion time of job i

when the speeds $\mathbf{s}^\alpha$ are used and $\bar{C}_i^\alpha$ when the rounded ones $\bar{\mathbf{s}}^\alpha$ are used. Since the speeds are reduced at most by $(1+\delta)$, then $(1+\delta)\bar{s_i}^\alpha \geq s_i^\alpha$, and we have that

$$\bar{C}_i^\alpha = \max\{r_i, \bar{C}_{i-1}^\alpha\} + \frac{\rho}{\bar{s}_i^\alpha} \leq (1+\delta)\left(\max\{r_i, C_{i-1}^\alpha\} + \frac{\rho}{s_i^\alpha}\right) = (1+\delta)C_i^\alpha, \tag{3.22}$$

which implies that $\sum_{i=1}^n w_i\bar{C}_i^\alpha \leq (1+\delta)\sum_{i=1}^n w_i C_i^\alpha$ and proves the lemma. □

Finally, in steps 5 and 6 we compute the completion times with the rounded speeds and return the approximate schedule.

In the following subsections we show that the SAPAS algorithm returns a feasible schedule and we compute the approximation factors for different energy aware scheduling problems.

3.1.5.2 Single Machine Problem with No Constraints

In this subsection we consider the unconstrained single machine problem or $1||\sum E_i(s_i)+w_iC_i$, with $E_i(s_i) = v_i\rho_i s_i^{\beta-1}$. Since there are no release date constraints, it follows that there will be no idle time between jobs and therefore

$$C_{\pi(i)} = \sum_{j=1}^{i} \frac{\rho_{\pi(j)}}{s_{\pi(j)}}. \tag{3.23}$$

Also note that because there are no precedence constraints the schedule given by the SAPAS algorithm is always feasible. The following is the main result for this setting.

Theorem 3.2. *The SAPAS algorithm with $\alpha = \frac{1}{2}$ returns a result that is at most $4(1+\delta)$ times the optimal value for the time-and-speed-indexed integer formulation for the $1||\sum E_i(s_i) + w_iC_i$ problem, with $E_i(s_i) = v_i\rho_i s_i^{\beta-1}$.*

Proof. W.l.o.g. we assume that $t_1^\alpha \leq t_2^\alpha \leq \ldots \leq t_n^\alpha$. Let y_{ijt}^* denote an optimal solution of the integer program (3.12)-(3.16), $\bar{y}_{ijt}$ the fractional solution of its linear relaxation, and $\tilde{y}_{ijt}$ the auxiliary variables calculated for the SAPAS algorithm.

First, because $\bar{\mathbf{y}}$ is a fractional solution we have that

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{t=1}^{T}\left(v_i\rho_i\sigma_j^{\beta-1} + w_i\tau t\right)\bar{y}_{ijt} \;\leq\; \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{t=1}^{T}\left(v_i\rho_i\sigma_j^{\beta-1} + w_i\tau t\right)y_{ijt}^*. \tag{3.24}$$

The energy terms of the algorithm's solution are bounded as follows,

$$\begin{aligned} v_i\rho_i(s_i^\alpha)^{\beta-1} &= v_i\rho_i\left(\frac{1}{s_i^\alpha}\right)^{-(\beta-1)} = v_i\rho_i\left(\mathbb{E}\left[\frac{1}{\hat{s}_i}\right]\right)^{-(\beta-1)} \\ &\leq v_i\rho_i\mathbb{E}\left[\left(\frac{1}{\hat{s}_i}\right)^{-(\beta-1)}\right] = v_i\rho_i\mathbb{E}\left[\hat{s}_i^{\beta-1}\right] = v_i\rho_i\sum_{j=1}^{m}\mu_{ij}\sigma_j^{\beta-1}, \end{aligned} \tag{3.25}$$

where the inequality follows from Jensen's Inequality applied to the convex function $\frac{1}{s^{\beta-1}}$. Using the definition of μ_{ij} it follows that,

$$v_i\rho_i(s_i^\alpha)^{\beta-1} \leq \frac{v_i\rho_i}{\alpha}\sum_{j=1}^{m}\sum_{u=1}^{t_i^\alpha}\sigma_j^{\beta-1}\tilde{y}_{iju} \leq \frac{v_i\rho_i}{\alpha(1-\alpha)}\sum_{j=1}^{m}\sum_{u=1}^{T}\sigma_j^{\beta-1}\bar{y}_{iju}, \tag{3.26}$$

where the last inequality comes from the fact that $\alpha < 1$ and $\tilde{y}_{ijt} \leq \bar{y}_{ijt}$.

The completion time terms are bounded as follows. Since there are no release dates there is no idle time between jobs, therefore, the completion time of job i is $C_i^\alpha = \sum_{j=1}^{i}\frac{\rho_j}{s_j^\alpha}$. From the definition of s_i^α we have,

$$\begin{aligned} C_i^\alpha &= \sum_{j=1}^{i}\frac{\rho_j}{s_j^\alpha} = \sum_{j=1}^{i}\rho_j\mathbb{E}\left[\frac{1}{\hat{s}_i}\right] = \sum_{j=1}^{i}\sum_{l=1}^{m}\rho_j\frac{\mu_{jl}}{\sigma_l} \\ &= \frac{1}{\alpha}\sum_{j=1}^{i}\sum_{l=1}^{m}\sum_{u=1}^{t_j^\alpha}\frac{\rho_j}{\sigma_l}\tilde{y}_{jlu} = \frac{\tau}{\alpha}\sum_{j=1}^{i}\sum_{l=1}^{m}\sum_{u=1}^{t_j^\alpha}\hat{p}_{jl}\tilde{y}_{jlu}. \end{aligned} \tag{3.27}$$

Additionally, summing up constraint (3.14) from $t = 1$ to $t = t_i^\alpha$, we have that

$$\sum_{j=1}^{n}\sum_{l=1}^{m}\sum_{u=t}^{t+\hat{p}_{jl}-1}\tilde{y}_{jlu} \leq 1 \Rightarrow \sum_{j=1}^{n}\sum_{l=1}^{m}\sum_{t=1}^{t_i^\alpha}\sum_{u=t}^{t+\hat{p}_{jl}-1}\tilde{y}_{jlu} \leq t_i^\alpha. \tag{3.28}$$

By switching the last two sums and eliminating some terms it is easy to demonstrate that

$$\sum_{u=1}^{t_i^\alpha}\hat{p}_{jl}\tilde{y}_{jlu} \leq \sum_{t=1}^{t_i^\alpha}\sum_{u=t}^{t+\hat{p}_{jl}-1}\tilde{y}_{jlu},$$

and hence, we have that

$$\sum_{j=1}^{n}\sum_{l=1}^{m}\sum_{u=1}^{t_i^\alpha}\hat{p}_{jl}\tilde{y}_{jlu} \leq \sum_{j=1}^{n}\sum_{l=1}^{m}\sum_{t=1}^{t_i^\alpha}\sum_{u=t}^{t+\hat{p}_{jl}-1}\tilde{y}_{jlu} \leq t_i^\alpha. \tag{3.29}$$

Since $t_j^\alpha \leq t_i^\alpha$, $\forall j \leq i$, then by summing up to t_j^α instead of t_i^α in (3.29), we get

$$\sum_{j=1}^{i}\sum_{l=1}^{m}\sum_{u=1}^{t_j^\alpha}\hat{p}_{jl}\tilde{y}_{jlu} \leq \sum_{j=1}^{n}\sum_{l=1}^{m}\sum_{u=1}^{t_i^\alpha}\hat{p}_{jl}\tilde{y}_{jlu} \leq t_i^\alpha. \tag{3.30}$$

From (3.27) and (3.30), it follows that

$$C_i^\alpha = \sum_{j=1}^{i}\frac{\rho_j}{s_j^\alpha} = \frac{\tau}{\alpha}\sum_{j=1}^{i}\sum_{l=1}^{m}\sum_{u=1}^{t_j^\alpha}\hat{p}_{jl}\tilde{y}_{jlu} \leq \frac{\tau}{\alpha}t_i^\alpha. \tag{3.31}$$

Let the fractional completion time $\bar{C}_i$ of job i be defined by

$$\bar{C}_i = \tau\sum_{j=1}^{m}\sum_{t=1}^{T} t\bar{y}_{ijt}. \tag{3.32}$$

Because it is possible that $\sum_{j=1}^{m}\sum_{t=1}^{t_i^\alpha}\bar{y}_{ijt} > \alpha$, we define $Y_i^{(1)} = \alpha - \sum_{j=1}^{m}\sum_{t=1}^{t_i^\alpha-1}\bar{y}_{ijt}$ and $Y_i^{(2)} = \sum_{j=1}^{m}\sum_{t=1}^{t_i^\alpha}\bar{y}_{ijt} - \alpha$. Thus, $Y_i^{(1)} + Y_i^{(2)} = \sum_{j=1}^{m}\bar{y}_{ijt_i^\alpha}$, and we have that

$$\begin{aligned}\bar{C}_i &= \tau\sum_{j=1}^{m}\sum_{t=1}^{t_i^\alpha-1} t\bar{y}_{ijt} + \tau t_i^\alpha\sum_{j=1}^{m}\bar{y}_{ijt_i^\alpha} + \tau\sum_{j=1}^{m}\sum_{t=t_i^\alpha+1}^{T} t\bar{y}_{ijt} \\ &= \tau\sum_{j=1}^{m}\sum_{t=1}^{t_i^\alpha-1} t\bar{y}_{ijt} + \tau t_i^\alpha Y_i^{(1)} + \tau t_i^\alpha Y_i^{(2)} + \tau\sum_{j=1}^{m}\sum_{t=t_i^\alpha+1}^{T} t\bar{y}_{ijt}.\end{aligned} \tag{3.33}$$

Eliminating the lower terms of the sum in (3.33) it follows that

$$\bar{C}_i \geq \tau t_i^\alpha Y_i^{(2)} + \tau\sum_{j=1}^{m}\sum_{t=t_i^\alpha+1}^{T} t\bar{y}_{ijt} \geq \tau t_i^\alpha Y_i^{(2)} + \tau\sum_{j=1}^{m} t_i^\alpha\bar{y}_{ijt} = \tau t_i^\alpha(1-\alpha), \tag{3.34}$$

and from (3.31) and (3.34) we get that

$$C_i^\alpha \leq \frac{\bar{C}_i}{\alpha(1-\alpha)}, \tag{3.35}$$

and thus,

$$\sum_{i=1}^{n} w_i C_i^\alpha \leq \frac{1}{\alpha(1-\alpha)}\sum_{i=1}^{n} w_i\bar{C}_i = \frac{1}{\alpha(1-\alpha)}\sum_{i=1}^{n} w_i\tau\sum_{j=1}^{m}\sum_{t=1}^{T} t\bar{y}_{ijt}. \tag{3.36}$$

From (3.26) and (3.36), it follows that

$$\sum_{i=1}^{n} v_i \rho_i (s_i^{\alpha})^{\beta-1} + \sum_{i=1}^{n} w_i C_i^{\alpha} \leq \frac{1}{\alpha(1-\alpha)} \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{t=1}^{T} \left(v_i \rho_i \sigma_j^{\beta-1} + w_i \tau t \right) \bar{y}_{ijt} \tag{3.37}$$

$$\leq \frac{1}{\alpha(1-\alpha)} \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{t=1}^{T} \left(v_i \rho_i \sigma_j^{\beta-1} + w_i \tau t \right) y^*_{ijt}, \tag{3.38}$$

where the last inequality comes from (3.24). Finally, by taking $\alpha = \frac{1}{2}$, which minimizes the R.H.S. of (3.38), and by Lemma 3.3 we complete the proof. □

3.1.5.3 Single Machine Problem with Precedence Constraints

First we analyze the case with only precedence constraints. Through the following lemma, we can prove that the SAPAS algorithm returns a feasible schedule for this problem.

Lemma 3.4. *If $i_1 \prec i_2$, constraint (3.17) implies that $t_{i_1}^{\alpha} < t_{i_2}^{\alpha}$.*

Proof. Given a precedence constraint $i_1 \prec i_2$, and using its corresponding constraint in the LP, given by equation (3.17), we have that for $t = \{1, \ldots, T - \hat{p}_{i_2 m}\}$,

$$\sum_{j=1}^{m} \sum_{u=1}^{t} \bar{y}_{i_1 j u} \geq \sum_{j=1}^{m} \sum_{u=1}^{\min\{t+\hat{p}_{i_2 j}, T\}} \bar{y}_{i_2 j u} \geq \sum_{j=1}^{m} \sum_{u=1}^{t+\hat{p}_{i_2 m}} \bar{y}_{i_2 j u}. \tag{3.39}$$

Taking $t = t_{i_2}^{\alpha} - \hat{p}_{i_2 m} \leq T - \hat{p}_{i_2 m}$, we get that

$$\sum_{j=1}^{m} \sum_{u=1}^{t_{i_2}^{\alpha} - \hat{p}_{i_2 m}} \bar{y}_{i_1 j u} \geq \sum_{j=1}^{m} \sum_{u=1}^{t_{i_2}^{\alpha}} \bar{y}_{i_2 j u} \geq \alpha, \tag{3.40}$$

where the last inequality follows from the definition of $t_{i_2}^{\alpha}$ given in (3.18). Equation (3.40) also implies that by time step $t_{i_2}^{\alpha} - \hat{p}_{i_2 m}$ job i_1 has already achieved its α-point so,

$$t_{i_1}^{\alpha} \leq t_{i_2}^{\alpha} - \hat{p}_{i_2 m} \Rightarrow t_{i_1}^{\alpha} < t_{i_2}^{\alpha}, \tag{3.41}$$

where the last inequality follows from $\hat{p}_{i_2,m} > 0$. □

Now we are in position of computing an approximation bound for the SAPAS algorithm in the $1|prec|\sum E_i(s_i) + w_iC_i$ case.

Theorem 3.3. *The SAPAS algorithm with $\alpha = \frac{1}{2}$ returns a result that is at most $4(1+\delta)$ times the optimal value for the time-and-speed-indexed integer formulation for the $1|prec|\sum E_i(s_i) + w_iC_i$ problem, with $E_i(s_i) = v_i\rho_i s_i^{\beta-1}$.*

Proof. Because the SAPAS algorithm sorts jobs in non-decreasing values of t_i^α, then if $t_{i_1}^\alpha < t_{i_2}^\alpha$, job i_1 is scheduled before job i_2. By Lemma 3.4, we get that the precedence constraint $i_1 \prec i_2$ implies $t_{i_1}^\alpha < t_{i_2}^\alpha$. Hence, the SAPAS algorithm preserves all the precedence constraints, and thus the resulting schedule is feasible.

Given this result, Theorem 3.2 proves that the SAPAS algorithm is a $4(1+\delta)$-approximation algorithm for the $1|prec|\sum E_i(s_i) + w_iC_i$ problem. □

3.1.5.4 Single Machine Problem with Precedence and Release Date Constraints

When release date constraints are added we get the following result.

Theorem 3.4. *The SAPAS algorithm with $\alpha = \sqrt{2} - 1$ returns a result that is at most $(3 + 2\sqrt{2})(1 + \delta)$ times the optimal value for the time-and-speed-indexed integer formulation for the $1|r_i, prec|\sum E_i(s_i) + w_iC_i$ problem, with $E_i(s_i) = v_i\rho_i s_i^{\beta-1}$*

Proof. We assume, w.l.o.g. that $t_1^\alpha \leq t_2^\alpha \leq \ldots \leq t_n^\alpha$. Because equation (3.26) from Theorem 3.2 is still valid, we have that

$$v_i\rho_i(s_i^\alpha)^{\beta-1} \leq \frac{v_i\rho_i}{\alpha(1-\alpha)} \sum_{j=1}^{m}\sum_{u=1}^{T} \sigma_j^{\beta-1}\bar{y}_{iju} \leq \frac{(1+\alpha)}{\alpha(1-\alpha)} v_i\rho_i \sum_{j=1}^{m}\sum_{u=1}^{T} \sigma_j^{\beta-1}\bar{y}_{iju}, \tag{3.42}$$

where the last inequality comes from the fact that $\alpha \geq 0$.

Since in this case idle times between jobs can exist, equation (3.27) is no longer valid. Because job i is scheduled to start after jobs 1 to $i-1$, it must start after $\max_{j=1}^{i} r_j$. Additionally, it will require at most $\sum_{j=1}^{i} \frac{\rho_j}{s_j^\alpha}$ time to be processed, thus, we can bound C_i^α by

$$C_i^\alpha \leq \max_{j=\{1,\ldots,i\}} r_j + \sum_{j=1}^{i} \frac{\rho_j}{s_j^\alpha}. \tag{3.43}$$

Job i has to start after its release date and the release date of all previous jobs, thus, $\max_{j=1}^{i} r_j \leq \tau t_i^\alpha$. From equation (3.31) we can bound $\sum_{j=1}^{i} \frac{\rho_j}{s_j^\alpha}$, hence,

$$C_i^\alpha \leq \tau t_i^\alpha + \frac{\tau}{\alpha} t_i^\alpha = \tau t_i^\alpha \left(1 + \frac{1}{\alpha}\right). \tag{3.44}$$

Since the equation (3.34) is still valid, together with (3.44) we get that

$$C_i^\alpha \leq \frac{(1+\alpha)}{\alpha(1-\alpha)} \bar{C}_i \Rightarrow \sum_{i=1}^{n} w_i C_i^\alpha \leq \frac{(1+\alpha)}{\alpha(1-\alpha)} \sum_{i=1}^{n} w_i \bar{C}_i. \tag{3.45}$$

From (3.42) and (3.45) it follows that,

$$\sum_{i=1}^{n} v_i \rho_i (s_i^\alpha)^{\beta-1} + \sum_{i=1}^{n} w_i C_i^\alpha \leq \frac{(1+\alpha)}{\alpha(1-\alpha)} \left[\sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{t=1}^{T} v_i \rho_i \sigma_j^{\beta-1} \bar{y}_{ijt} + \sum_{i=1}^{n} w_i \bar{C}_i \right] \tag{3.46}$$

$$\leq \frac{(1+\alpha)}{\alpha(1-\alpha)} \left[\sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{t=1}^{T} v_i \rho_i \sigma_j^{\beta-1} y_{ijt}^* + \sum_{i=1}^{n} w_i C_i^* \right] \tag{3.47}$$

Finally, by setting $\alpha = \arg\min_{\{0 \leq \alpha \leq 1\}} \left\{ \frac{(1+\alpha)}{\alpha(1-\alpha)} \right\} = \sqrt{2} - 1$, and by Lemma 3.3 it follows that the SAPAS algorithm is a $(3 + 2\sqrt{2})(1 + \delta)$-approximation algorithm. □

Note that for these last two cases we still have the same upper and lower bounds on the possible speed values $\sigma_{\max}$ and $\sigma_{\min}$, as well as the bound used for the total number of time-steps, T.

3.1.6 Interval-and-Speed-Indexed Formulation

In the interval-indexed formulation we divide the time horizon into geometrically increasing intervals, and the completion time of each job is assigned to one of these intervals. Since the completion times are not associated to a specific time, the completion times are not precisely known but are lower bounded. By controlling the growth of each interval one can obtain a sufficiently tight bound.

3.1.6.1 Model Description and Approximation Algorithm

The problem formulation is as follows. We divide the time horizon into the following geometrically increasing intervals: $[\kappa, \kappa]$, $(\kappa, (1+\epsilon)\kappa]$, $((1+\epsilon)\kappa, (1+\epsilon)^2\kappa]$, ..., where $\epsilon > 0$ is an arbitrary small constant, and $\kappa = \frac{\rho_{\min}}{\sigma_{\max}}$ denotes the smallest interval size that will hold at least one whole job. We define interval $I_t = (\tau_{t-1}, \tau_t]$, with $\tau_0 = \kappa$ and $\tau_t = \kappa(1+\epsilon)^{t-1}$. The interval index ranges over $\{1, \ldots, T\}$, with $T = \min\{\lceil t \rceil : \kappa(1+\epsilon)^{t-1} \geq \max_{i=1}^{n} r_i + \sum_{i=1}^{n} \frac{\rho_i}{\sigma_1}\}$; and thus, we have a polynomial number of indices.

Let

$$x_{ijt} = \begin{cases} 1, & \text{if job } i \text{ runs at a speed } \sigma_j \text{ and completes in1 time interval } I_t = (\tau_{t-1}, \tau_t] \\ 0, & \text{otherwise.} \end{cases} \tag{3.48}$$

By using the lower bounds τ_{t-1} of each time interval I_t, a lower bound to (3.1) is written as,

$$\min_{\mathbf{x}} \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{t=1}^{T} \left(v_i \rho_i \sigma_j^{\beta-1} + w_i \tau_{t-1} \right) x_{ijt}. \tag{3.49}$$

The following are the constraints required for the $1|r_i, prec| \sum E_i(s_i) + w_i C_i$ problem:

1. Each job must finish in a unique time interval and speed; therefore for $i = \{1, \ldots, n\}$:

$$\sum_{j=1}^{m} \sum_{t=1}^{T} x_{ijt} = 1. \tag{3.50}$$

2. Since only one job can be processed at any given time, the total processing time of jobs up to time interval I_t must be at most τ_t units. Thus, for $t = \{1, \ldots, T\}$:

$$\sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{u=1}^{t} \frac{\rho_i}{\sigma_j} x_{iju} \leq \tau_t. \tag{3.51}$$

3. Job i running at speed σ_j requires $\frac{\rho_i}{\sigma_j}$ time units to be processed, and considering that its release time is r_i, then for $i = \{1, \ldots, n\}$, $j = \{1, \ldots, m\}$, and $t = \{1, \ldots, T\}$:

$$x_{ijt} = 0, \quad \text{if } \tau_t < r_i + \frac{\rho_i}{\sigma_j}. \tag{3.52}$$

4. For $i = \{1, \ldots, n\}$ and $t = \{1, \ldots, T\}$:

$$x_{it} \in \{0, 1\}. \tag{3.53}$$

5. The precedence constraint $i_1 \prec i_2$ implies that job i_2 cannot finish in an interval earlier than i_1. Therefore for every $i_1 \prec i_2$ constraint we have that for $t = \{1, \ldots, T\}$:

$$\sum_{j=1}^{m} \sum_{u=1}^{t} x_{i_1 j u} \geq \sum_{j=1}^{m} \sum_{u=1}^{t} x_{i_2 j u}. \tag{3.54}$$

It is important to note that this integer program only provides a lower bound for (3.1); in fact its optimal solution may not be schedulable, since constraints (3.51) do not imply that only one job can be processed at a single time, they only bound the total amount of work in $\cup_t I_t$. Hence, for example, an instance in which all the release dates of the jobs assigned to a specific interval $I_t = (\tau_{t-1}, \tau_t]$ are at $r_i = \tau_t - \delta$, with δ arbitrary small, will be feasible in the IP as long as the processing time of all jobs assigned to I_t is less than $(1 + \epsilon)\tau_{t-1}$. Because of the release dates, if the total processing time of the jobs is more than δ, the jobs will not fit in the interval, making it not schedulable, even though it is feasible in the IP.

We now describe the approximation algorithm for the EAS problem with weighted completion time as scheduling metric, called Schedule by α-intervals and α-speeds (SAIAS), which is detailed in Algorithm 3.2.

Let $\bar{x}_{ijt}$ denote the optimal solution of the linear relaxation of the integer program (3.49)-(3.54), in which we change constraints (3.53) for $x_{ijt} \geq 0$. In step 1 we formulate the IP and in step 2 of the algorithm we compute the optimal solution $\bar{\mathbf{x}}$. Next, in step 3, given $0 \leq \alpha \leq 1$, we compute the α-interval of job i, which is defined as,

$$I_i^{\alpha} = \min \left\{ t : \sum_{j=1}^{m} \sum_{u=1}^{t} \bar{x}_{iju} \geq \alpha \right\}. \tag{3.55}$$

Since several jobs may finish in the same interval, let J_t denote the set of jobs that finish in interval I_t, $J_t = \{i : I_i^{\alpha} = t\}$, and we use these sets to determine the order Π^{α} as described in step 4.

Algorithm 3.2 SCHEDULE BY α-INTERVALS AND α-SPEEDS (SAIAS) FOR EAS

Inputs: set of jobs, $\alpha \in (0,1)$, $\epsilon > 0$, set of speeds $\mathbf{S} = \{\sigma_1, \ldots, \sigma_m\}$.

1 Divide time into increasing time intervals $I_t = (\tau_{t-1}, \tau_t]$, with $\tau_t = \kappa(1+\epsilon)^{t-1}$.

2 Compute an optimal solution $\bar{\mathbf{x}}$ to the linear relaxation (3.49)-(3.54).

3 Compute the α-intervals $\mathbf{I}^\alpha$ and the sets J_t.

4 Compute an order Π^α that has the sets J_t ordered in non-decreasing values of t and the jobs within each set in a manner consistent with the precedence constraints.

5 Compute the α-speeds $\mathbf{s}^\alpha$.

6 Round down each s_i^α to the nearest speed in $\mathbf{S}$ and run job i at this rounded speed, $\bar{s}_i^\alpha$.

7 Set the i-th job to start at time $\max\{r_{\pi(i)}, \bar{C}^\alpha_{\pi(i-1)}\}$, where $\bar{C}^\alpha_{\pi(i-1)}$ is the completion time of the previous job using the rounded α-speeds, and $\bar{C}^\alpha_{\pi(0)} = 0$.

8 **return** speeds $\bar{\mathbf{s}}^\alpha$, order Π^α and completion times $\bar{\mathbf{C}}^\alpha$.

In step 5, we compute the α-speeds as follows. Since $\sum_{j=1}^m \sum_{u=1}^{I_i^\alpha} \bar{x}_{iju} \geq \alpha$, we define auxiliary variable $\{\tilde{x}_{ijt}\}$ as:

$$\tilde{x}_{ijt} = \begin{cases} \bar{x}_{ijt}, & t < I_i^\alpha \\ \max\left\{\min\left\{\bar{x}_{ijI_i^\alpha}, \alpha - \sum_{l=1}^{j-1} \bar{x}_{ilI_i^\alpha} - \beta_i\right\}, 0\right\}, & t = I_i^\alpha \\ 0, & t > I_i^\alpha, \end{cases} \tag{3.56}$$

where $\beta_i = \sum_{j=1}^m \sum_{u=1}^{I_i^\alpha - 1} \bar{x}_{iju} < \alpha$. Note that with this auxiliary variable we have that

$$\sum_{j=1}^{m} \sum_{u=1}^{I_i^\alpha} \tilde{x}_{iju} = \alpha.$$

This is a key step that allows us to truncate the fractional solution so that for every job i, the sum of $\tilde{x}_{ijt}$ up to time interval I_i^α for each speed j can be interpreted as a probability mass function. We define this probability mass function (pmf) $\mu_i = (\mu_{i1}, \ldots, \mu_{im})$ on the set of speeds $\mathbf{S} = \{\sigma_1, \ldots, \sigma_m\}$ as

$$\mu_{ij} = \frac{1}{\alpha} \sum_{u=1}^{I_i^\alpha} \tilde{x}_{iju}. \tag{3.57}$$

Let $\hat{s}_i$ define a random variable distributed according to the pmf μ_i, i.e. $\mu_{ij} = \mathbb{P}(\hat{s}_i = \sigma_j)$. Then, the α-speed of job i, s_i^α, is defined as follows:

$$\frac{1}{s_i^\alpha} = \mathbb{E}\left[\frac{1}{\hat{s}_i}\right] = \sum_{j=1}^{m} \frac{\mu_{ij}}{\sigma_j} \Rightarrow s_i^\alpha = \frac{1}{\mathbb{E}\left[\frac{1}{\hat{s}_i}\right]}. \tag{3.58}$$

We define the α-speeds using the reciprocal of the speeds since the completion times are proportional to the reciprocals instead of the speeds, and we need to bound completion times in the analysis of the algorithm.

Next, in step 6, because the α-speeds s_i^α do not necessarily belong to the set of possible speeds $\mathbf{S}$ we round them down to $\bar{s}_i^\alpha$, which is the nearest speed in the set such that $\bar{s}_i^\alpha \leq s_i^\alpha$. Finally, in steps 7 and 8 we compute the completion times given the calculated speeds and return the set of speeds $\bar{\mathbf{s}}^\alpha$ and the schedule given by the order Π^α and the completion times $\bar{\mathbf{C}}^\alpha$.

We now analyse this algorithm's performance for different energy aware scheduling problems. In the following subsections we will assume w.l.o.g. that $I_1^\alpha \leq I_2^\alpha \leq \ldots I_n^\alpha$.

3.1.6.2 Single Machine Problem with Precedence Constraints

We first need to prove that the output of the SAIAS algorithm is indeed feasible.

Lemma 3.5. *If $i_1 \prec i_2$, then constraint (3.54) implies that $I_{i_1}^\alpha \leq I_{i_2}^\alpha$.*

Proof. Evaluating the LP constraint (3.54) corresponding to $i_1 \prec i_2$, for $t = I_{i_2}^\alpha$, we have that,

$$\sum_{j=1}^{m} \sum_{u=1}^{I_{i_2}^\alpha} x_{i_1 j u} \geq \sum_{j=1}^{m} \sum_{u=1}^{I_{i_2}^\alpha} x_{i_2 j u} \geq \alpha,$$

where the last inequality follows from the definition of $I_{i_2}^\alpha$. The chain of inequalities implies that

$$\sum_{j=1}^{m} \sum_{u=1}^{I_{i_2}^\alpha} x_{i_1 j u} \geq \alpha,$$

so $I_{i_1}^\alpha \leq I_{i_2}^\alpha$. □

Since the SAIAS algorithm schedules jobs by first ordering the sets J_t in increasing order of t, and then orders the jobs within each set in a way that is consistent with the precedence constraints, by Lemma 3.5 it follows that the SAIAS algorithm preserves the precedence constraints, and, therefore, the output of the algorithm is feasible. Next, we can prove the following result.

Theorem 3.5. *The SAIAS algorithm with $\alpha = \frac{1}{2}$ is a $4(1+\epsilon)(1+\delta)$-approximation algorithm for the $1|prec|\sum E_i(s_i) + w_iC_i$ problem, with $E_i(s_i) = v_i\rho_i s_i^{\beta-1}$.*

Proof. Let x^*_{ijt} denote an optimal solution to the integer problem (3.49)-(3.54), $\bar{x}_{ijt}$ the fractional solution of its linear relaxation, and $\tilde{x}_{iju}$ the auxiliary variables calculated for the SAIAS algorithm.

Since in (3.49) the completion time for jobs completed in interval I_t is τ_{t-1}, it follows that,

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{t=1}^{T}\left(v_i\rho_i\sigma_j^{\beta-1} + w_i\tau_{t-1}\right)\bar{x}_{ijt} \leq \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{t=1}^{T} v_i\rho_i\sigma_j^{\beta-1}x^*_{ijt} + \sum_{i=1}^{n} w_iC^*_i. \tag{3.59}$$

The energy terms of the algorithm's solution are bounded as follows,

$$\begin{aligned} v_i\rho_i(s_i^{\alpha})^{\beta-1} &= v_i\rho_i\left(\frac{1}{s_i^{\alpha}}\right)^{-(\beta-1)} = v_i\rho_i\left(\mathbb{E}\left[\frac{1}{\hat{s}_i}\right]\right)^{-(\beta-1)} \\ &\leq v_i\rho_i\mathbb{E}\left[\left(\frac{1}{\hat{s}_i}\right)^{-(\beta-1)}\right] = v_i\rho_i\mathbb{E}\left[\hat{s}_i^{\beta-1}\right] = v_i\rho_i\sum_{j=1}^{m}\mu_{ij}\sigma_j^{\beta-1}, \end{aligned} \tag{3.60}$$

where the inequality follows from Jensen's Inequality applied to the convex function $\frac{1}{s^{\beta-1}}$. Using the definition of μ_{ij} in (3.57) and given that $0 \leq \alpha \leq 1$, $\epsilon > 0$, and $\tilde{x}_{ijt} \leq \bar{x}_{ijt}$, it follows that,

$$v_i\rho_i(s_i^{\alpha})^{\beta-1} \leq \frac{v_i\rho_i}{\alpha}\sum_{j=1}^{m}\sum_{u=1}^{I_i^{\alpha}}\sigma_j^{\beta-1}\tilde{x}_{iju} \leq \frac{(1+\epsilon)}{\alpha(1-\alpha)}v_i\rho_i\sum_{j=1}^{m}\sum_{u=1}^{T}\sigma_j^{\beta-1}\bar{x}_{iju}. \tag{3.61}$$

Since there are no release date constraints there is no idle time between jobs,

$$C_i^{\alpha} = \sum_{j=1}^{i}\frac{\rho_j}{s_j^{\alpha}} = \sum_{j=1}^{i}\rho_j\mathbb{E}\left[\frac{1}{\hat{s}_j}\right] = \frac{1}{\alpha}\sum_{j=1}^{i}\sum_{l=1}^{m}\sum_{u=1}^{I_j^{\alpha}}\frac{\rho_j}{\sigma_l}\tilde{x}_{jlu} \leq \frac{1}{\alpha}\sum_{j=1}^{n}\sum_{l=1}^{m}\sum_{u=1}^{I_i^{\alpha}}\frac{\rho_j}{\sigma_l}\bar{x}_{jlu}, \tag{3.62}$$

and from constraint (3.51) for $t = I_i^\alpha$ we get, $C_i^\alpha \leq \frac{1}{\alpha}\tau_{I_i^\alpha}$.

Let $\bar{C}_i = \sum_{j=1}^{m}\sum_{t=1}^{T}\tau_{t-1}\bar{x}_{ijt}$ denote the optimal fractional completion time given by the optimal solution of the relaxed linear program (3.49)-(3.52). Since it is possible that $\sum_{j=1}^{m}\sum_{t=1}^{I_i^\alpha}\bar{x}_{ijt} > \alpha$; we define $X_i^{(1)} = \alpha - \sum_{j=1}^{m}\sum_{t=1}^{I_i^\alpha-1}\bar{x}_{ijt}$ and $X_i^{(2)} = \sum_{j=1}^{m}\sum_{t=1}^{I_i^\alpha}\bar{x}_{ijt} - \alpha$, thus $X_i^{(1)} + X_i^{(2)} = \sum_{j=1}^{m}\bar{x}_{ijI_i^\alpha}$, and we can rewrite

$$\bar{C}_i = \sum_{j=1}^{m}\sum_{t=1}^{I_i^\alpha-1}\tau_{t-1}\bar{x}_{ijt} + \tau_{I_i^\alpha-1}X_i^{(1)} + \tau_{I_i^\alpha-1}X_i^{(2)} + \sum_{j=1}^{m}\sum_{t=I_i^\alpha+1}^{T}\tau_{t-1}\bar{x}_{ijt}, \tag{3.63}$$

and eliminating the lower terms of the previous sum we get that,

$$\begin{aligned}\bar{C}_i &\geq \tau_{I_i^\alpha-1}X_i^{(2)} + \sum_{j=1}^{m}\sum_{t=I_i^\alpha+1}^{T}\tau_{t-1}\bar{x}_{ijt} \\ &\geq \tau_{I_i^\alpha-1}X_i^{(2)} + \sum_{j=1}^{m}\sum_{t=I_i^\alpha+1}^{T}\tau_{I_i^\alpha-1}\bar{x}_{ijt} = \tau_{I_i^\alpha-1}(1-\alpha).\end{aligned} \tag{3.64}$$

Because $\tau_{I_i^\alpha} = (1+\epsilon)\tau_{I_i^\alpha-1}$, from (3.62) and (3.64) we get that

$$C_i^\alpha \leq \frac{(1+\epsilon)}{\alpha(1-\alpha)}\bar{C}_i \Rightarrow \sum_{i=1}^{n} w_i C_i^\alpha \leq \frac{(1+\epsilon)}{\alpha(1-\alpha)}\sum_{i=1}^{n} w_i\bar{C}_i. \tag{3.65}$$

From (3.59), (3.61), and (3.65) it follows that,

$$\sum_{i=1}^{n} v_i\rho_i(s_i^\alpha)^{\beta-1} + \sum_{i=1}^{n} w_i C_i^\alpha \leq \frac{(1+\epsilon)}{\alpha(1-\alpha)}\left[\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{t=1}^{T} v_i\rho_i\sigma_j^{\beta-1}x_{ijt}^* + \sum_{i=1}^{n} w_i C_i^*\right], \tag{3.66}$$

and we set $\alpha = \arg\min_{0\leq\alpha\leq1}\left\{\frac{1}{\alpha(1-\alpha)}\right\} = \frac{1}{2}$, to minimize the bound. By Lemma 3.3, which bounds the final rounding error, we get the desired approximation ratio. □

3.1.6.3 Single Machine Problem with Precedence and Release Date Constraints

We now analyse the case with precedence constraints and release dates. Release dates makes the problem somewhat harder since they can introduce idle times between jobs.

Theorem 3.6. *The SAIAS algorithm with* $\alpha = \sqrt{2} - 1$ *is a* $(3 + 2\sqrt{2})(1+\epsilon)(1+\delta)$*-approximation algorithm for the* $1|r_i, prec|\sum E_i(s_i) + w_i C_i$ *problem, with* $E_i(s_i) = v_i\rho_i s_i^{\beta-1}$.

Proof. The bound for the energy terms computed in equation (3.60) is still valid when there is idle time between jobs, thus, we have that,

$$v_i\rho_i(s_i^\alpha)^{\beta-1} \leq \frac{(1+\epsilon)}{\alpha(1-\alpha)} v_i\rho_i \sum_{j=1}^{m}\sum_{u=1}^{T} \sigma_j^{\beta-1}\bar{x}_{iju} \leq \frac{(1+\epsilon)(1+\alpha)}{\alpha(1-\alpha)} v_i\rho_i \sum_{j=1}^{m}\sum_{u=1}^{T} \sigma_j^{\beta-1}\bar{x}_{iju}. \tag{3.67}$$

When bounding the completion time C_i^α, given the sorting done in step 3 of the SAIAS algorithm, now one has to consider all the jobs up to the ones in set $J_{I_i^\alpha}$, and thus,

$$C_i^\alpha \leq \max_{j\in\{J_1,\ldots,J_{I_i^\alpha}\}} r_j + \sum_{j\in\{J_1,\ldots,J_{I_i^\alpha}\}} \frac{\rho_j}{s_j^\alpha}. \tag{3.68}$$

Since all jobs that have been at least partially processed up to time interval I_t need to be released before τ_t, it follows that $\max_{j\in\{J_1,\ldots,J_{I_i^\alpha}\}} r_j \leq \tau_{I_i^\alpha}$. On the other hand, we also have that,

$$\sum_{j\in\{J_1,\ldots,J_{I_i^\alpha}\}} \frac{\rho_j}{s_j^\alpha} = \frac{1}{\alpha} \sum_{j\in\{J_1,\ldots,J_{I_i^\alpha}\}} \sum_{l=1}^{m}\sum_{u=1}^{I_j^\alpha} \frac{\rho_j}{\sigma_l}\tilde{x}_{jlu} \leq \frac{1}{\alpha}\sum_{j=1}^{n}\sum_{l=1}^{m}\sum_{u=1}^{I_i^\alpha} \frac{\rho_j}{\sigma_l}\bar{x}_{jlu} \leq \frac{1}{\alpha}\tau_{I_i^\alpha}, \tag{3.69}$$

where the last inequality follows from constraint (3.51) with $t = I_i^\alpha$. Thus,

$$C_i^\alpha \leq \frac{(1+\alpha)}{\alpha}\tau_{I_i^\alpha}.$$

Since $\bar{C}_i = \sum_{j=1}^{m}\sum_{t=1}^{T} \tau_{t-1}\bar{x}_{ijt}$, (3.64) is still valid and because $\tau_{I_i^\alpha} = (1+\epsilon)\tau_{I_i^\alpha-1}$, we get that

$$C_i^\alpha \leq \frac{(1+\epsilon)(1+\alpha)}{\alpha(1-\alpha)}\bar{C}_i \Rightarrow \sum_{i=1}^{n} w_i C_i^\alpha \leq \frac{(1+\epsilon)(1+\alpha)}{\alpha(1-\alpha)} \sum_{i=1}^{n} w_i\bar{C}_i. \tag{3.70}$$

Finally, from (3.67) and (3.70) it follows that,

$$\sum_{i=1}^{n} v_i\rho_i(s_i^\alpha)^{\beta-1} + \sum_{i=1}^{n} w_i C_i^\alpha \leq \frac{(1+\epsilon)(1+\alpha)}{\alpha(1-\alpha)} \left[\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{t=1}^{T} v_i\rho_i\sigma_j^{\beta-1}x_{ijt}^* + \sum_{i=1}^{n} w_i C_i^*\right], \tag{3.71}$$

and by setting $\alpha = \arg\min_{0\leq\alpha\leq 1}\left\{\frac{(1+\alpha)}{\alpha(1-\alpha)}\right\} = \sqrt{2}-1$, and again using Lemma 3.3 to bound the speed-rounding error, we get the required approximation ratio. □

3.2 Total Weighted Tardiness

In this section we modify the SAIAS algorithm and apply it to the weighted tardiness setting. We still allow for arbitrary precedence constraints but no release dates. In this case, each job i also has a deadline d_i. The tardiness T_i of job i is defined as $T_i = \max\{0, C_i - d_i\}$

3.2.1 Problem Formulation

We have a set of n jobs, where each job i has the following characteristics: a starting time u_i, a completion time c_i, a workload or processing requirements $\rho_i > 0$, and it is run at a constant speed s_i. As we are not allowing preemption, we will have that $c_i = u_i + \frac{\rho_i}{s_i}$. Also each job might have an associated deadline d_i.

Additionally, using the vector of starting times $\mathbf{u}$, we can define a job ordering vector $\Pi = \{\pi(1), \ldots, \pi(n)\}$ that specifies the order in which the jobs will be processed.

Now, each job i will be processed on a single machine that consumes power according to the power function $P_i(s_i) = v_i s_i^{\beta}$, where $v_i \geq 0$ depends on the job but the exponent $\beta \geq 2$ does not. This means that the total energy cost by job i will be $E_i(s_i) = v_i s_i^{\beta} \frac{\rho_i}{s_i} = v_i \rho_i s_i^{\beta-1}$.

Our objective is then to find the best scheduling order Π with starting times $\mathbf{u}$ and speeds $\mathbf{s}$ that will minimize the total cost of completing all the jobs, i.e. it will minimize the function

$$g(\Pi, \mathbf{u}, \mathbf{s}) = \sum_{i=1}^{n} v_{\pi(i)} \rho_{\pi(i)} s_{\pi(i)}^{\beta-1} + w_{\pi(i)} (C_{\pi(i)} - d_{\pi(i)})^{+}.$$

Lemma 3.6. *For any given order* Π*, the optimal solution for* $\mathbf{s}$ *and* $\mathbf{u}$ *is such that* $u_{\pi(i)} = C_{\pi(i-1)}$*,* $\forall i \in \{1, \ldots, n\}$*, i.e. there is no idle time between jobs.*

Proof. By contradiction, let's assume there exists an order Π such that, for that order, the optimal solution $\mathbf{u}$ and $\mathbf{s}$ has idle time between at least one pair of jobs. Hence, $\exists j \in \{1, \ldots, n\}$ such that $u_{\pi(j)} > C_{\pi(j-1)}$.

Using $\mathbf{u}$, we can create a new starting times vector $\mathbf{u}'$ such that $u'_{\pi(j)} = C_{\pi(j-1)}$ and $u'_{\pi(i)} = u_{\pi(i)}$, $\forall i \neq j$, i.e. job $\pi(j)$ is released earlier. As $s_{\pi(i)} = \frac{\rho_{\pi(i)}}{C_{\pi(i)} - u_{\pi(i)}}$ we can define a new speed vector $\mathbf{s}'$ such that $s'_{\pi(i)} = \frac{\rho_{\pi(i)}}{C_{\pi(i)} - u'_{\pi(i)}}$, thus both solutions will have the same completion

times $\mathbf{C}$ for all jobs, and also only job $\pi(j)$ will have a different speed. Hence,

$$g(\Pi, \mathbf{u}, \mathbf{s}) - g(\Pi, \mathbf{u}', \mathbf{s}') = v_{\pi(i)}\rho_{\pi(i)}\left(s_{\pi(i)}^{\beta-1} - s'^{\beta-1}_{\pi(i)}\right) > 0 \Rightarrow g(\Pi, \mathbf{u}, \mathbf{s}) > g(\Pi, \mathbf{u}', \mathbf{s}'),$$

since $s_{\pi(i)} > s'_{\pi(i)}$ because $u_{\pi(j)} > u'_{\pi(j)}$, which is a contradiction, as that implies that $\mathbf{u}'$, $\mathbf{s}'$ is a better solution. □

Using Lemma 3.6 we get that the objective function can be written as

$$g(\Pi, \mathbf{s}) = \sum_{i=1}^{n} v_i\rho_i s_i^{\beta-1} + \sum_{i=1}^{n} w_{\pi(i)}\left(C_{\pi(i)} - d_{\pi(i)}\right)^+ \tag{3.72}$$

and we can formulate our problem as a combinatorial optimization one in $\mathbf{s}$ and Π, replacing u_i by C_{i-1}:

$$\min_{\Pi,\mathbf{s}} \sum_{i=1}^{n} v_{\pi(i)}\rho_{\pi(i)}s_{\pi(i)}^{\beta-1} + w_{\pi(i)}(C_{\pi(i)} - d_{\pi(i)})^+ \tag{3.73}$$

$$\text{s.t.:} \quad \begin{aligned} C_{\pi(i)} &= \textstyle\sum_{j=1}^{i} \frac{\rho_{\pi(j)}}{s_{\pi(j)}}, \quad \forall i \in \{1, \ldots, n\} \\ \mathbf{s} &\geq 0 \end{aligned}$$

3.2.2 Interval-and-Speed-Indexed Formulation

We now formulate the problem using a modification of the interval-and-speed-indexed formulation presented in Section 3.1.1. Because the completion time can be bounded by $\sum_{j=1}^{m}\sum_{t=1}^{T}\tau_{t-1}x_{ijt}$, we can bound (3.72) from below by the following optimization problem,

$$\min_{\mathbf{x}} \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{t=1}^{T}\left(v_i\rho_i\sigma_j^{\beta-1} + w_i\left(\tau_{t-1} - d_i\right)^+\right)x_{ijt}, \tag{3.74}$$

together with constraints (3.50)-(3.54) from the interval-indexed formulation. Note that although the objective (3.72) is non-linear, because we have a interval-indexed formulation, (3.74) is linear.

We compute an approximate optimal solution for (3.72) using the SCHEDULE BY α-INTERVALS AND α-SPEEDS FOR TARDINESS (SAIAS-T) Algorithm detailed in Algorithm 3.3. The main difference with the SAIAS algorithm, is that in step 5 we scale up the α-speeds.

Algorithm 3.3 SCHEDULE BY α-INTERVALS AND α-SPEEDS FOR TARDINESS (SAIAS-T)

Inputs: set of jobs, $\alpha \in (0,1)$, $\epsilon > 0$, $\gamma > 1$, set of speeds $\mathbf{S} = \{\sigma_1, \ldots, \sigma_m\}$.

1 Divide time into increasing time intervals $I_t = (\tau_{t-1}, \tau_t]$, with $\tau_t = \kappa(1+\epsilon)^{t-1}$.

2 Compute an optimal solution $\bar{\mathbf{x}}$ to the linear relaxation (3.74), (3.50)-(3.54).

3 Compute the α-intervals $\mathbf{I}^\alpha$ and the sets J_t as in the SAIAS algorithm.

4 Compute an order Π^α that has the sets J_t ordered in non-decreasing values of t and the jobs within each set in a manner consistent with the precedence constraints.

5 Compute the α-speeds $\mathbf{s}^\alpha$ and scale each s_i^α to $\tilde{s}_i^\alpha = \gamma s_i^\alpha$.

6 Round up each $\tilde{s}_i^\alpha$ to the next speed in $\mathbf{S}$, $\bar{s}_i^\alpha$ and run each job i at this new speed.

7 Set the i-th job to start at time $\max\{r_{\pi(i)}, \bar{C}^\alpha_{\pi(i-1)}\}$, where $\bar{C}^\alpha_{\pi(i-1)}$ is the completion time of the previous job using the rounded α-speeds, and $\bar{C}^\alpha_{\pi(0)} = 0$.

8 **return** speeds $\bar{\mathbf{s}}^\alpha$, order Π^α, and completion times $\bar{\mathbf{C}}^\alpha$.

This scaling makes the completion time of the relaxed LP comparable to the completion time of the algorithm's output, and thus jobs that have 0 tardiness in the LP also have 0 tardiness in our algorithm. If we rounded speeds down, jobs with 0 tardiness in the LP could, at a lower speed, miss their deadline, and thus the approximation ratio could be arbitrary large.

We now analyze the algorithm assuming w.l.o.g. that $I_1^\alpha \leq I_2^\alpha \leq \ldots \leq I_n^\alpha$. Since Lemma 3.5 remains valid, arguments identical to those in Section 3.1.6.2 show that the output of the SAIAS-T algorithm is feasible; thus, we have the following theorem:

Theorem 3.7. *The SAIAS-T algorithm with* $\gamma = \frac{(1+\epsilon)}{\alpha(1-\alpha)}$ *and* $\alpha = \frac{1}{2}$ *is a* $4^\beta(1+\epsilon)^{\beta-1}(1+\delta)^{\beta-1}$*-approximation algorithm for the* $1|prec|\sum E_i(s_i) + w_iT_i$ *problem, with* $E_i(s_i) = v_i\rho_i s_i^{\beta-1}$.

Proof. Let $\bar{C}_i = \sum_{j=1}^m \sum_{t=1}^T \tau_{t-1}\bar{x}_{ijt}$ denote the optimal fractional completion time of the relaxed linear program. $(\bar{C}_i - d_i)^+$ is a lower bound for the optimal tardiness $(C_i^* - d_i)^+$, since $\sum_{jt}(\tau_{t-1} - d_i)^+\bar{x}_{ijt} \geq (\bar{C}_i - d_i)^+$. Thus,

$$\sum_{i=1}^n \sum_{j=1}^m \sum_{t=1}^T v_i\rho_i\sigma_j^{\beta-1}\bar{x}_{ijt} + \sum_{i=1}^n w_i\left(\bar{C}_i - d_i\right)^+ \leq \sum_{i=1}^n \sum_{j=1}^m \sum_{t=1}^T v_i\rho_i\sigma_j^{\beta-1}x^*_{ijt} + \sum_{i=1}^n w_i\left(C_i^* - d_i\right)^+. \tag{3.75}$$

Let $\tilde{C}_i^\alpha$ denote the completion time of job i using speeds $\tilde{\mathbf{s}}^\alpha$ and C_i^α the one using speeds $\mathbf{s}^\alpha$. Because there are no release date constraints, there is no idle time in between jobs; therefore,

$$\tilde{C}_i^\alpha = \sum_{j=1}^{i} \frac{\rho_j}{\tilde{s}_j^\alpha} = \frac{1}{\gamma} \sum_{j=1}^{i} \frac{\rho_j}{s_j^\alpha} = \frac{1}{\gamma} C_i^\alpha. \tag{3.76}$$

Since (3.62) remains valid, it follows that

$$C_i^\alpha \leq \frac{(1+\epsilon)}{\alpha(1-\alpha)} \bar{C}_i \;\Rightarrow\; \tilde{C}_i^\alpha \leq \frac{1}{\gamma} \frac{(1+\epsilon)}{\alpha(1-\alpha)} \bar{C}_i.$$

The key step is that by setting $\gamma = \frac{(1+\epsilon)}{\alpha(1-\alpha)}$, which makes the two completion times comparable, we have that,

$$\sum_{i=1}^{n} w_i \left(\tilde{C}_i^\alpha - d_i\right)^+ \leq \sum_{i=1}^{n} w_i \left(\frac{1}{\gamma} \frac{(1+\epsilon)}{\alpha(1-\alpha)} \bar{C}_i - d_i\right)^+ = \sum_{i=1}^{n} w_i \left(\bar{C}_i - d_i\right)^+. \tag{3.77}$$

The energy term is bounded in a manner analogous to (3.61):

$$v_i \rho_i (\tilde{s}_i^\alpha)^{\beta-1} = \gamma^{\beta-1} v_i \rho_i (s_i^\alpha)^{\beta-1} \leq \frac{(1+\epsilon)^{\beta-1}}{(\alpha(1-\alpha))^\beta} v_i \rho_i \sum_{j=1}^{m} \sum_{t=1}^{T} \sigma_j^{\beta-1} \bar{x}_{ijt}, \tag{3.78}$$

where the last inequality follows from (3.61) that remains valid.

From (3.75), (3.78), and (3.77) it follows that,

$$\sum_{i=1}^{n} v_i \rho_i (\tilde{s}_i^\alpha)^{\beta-1} + \sum_{i=1}^{n} w_i \left(\tilde{C}_i^\alpha - d_i\right)^+ \leq \frac{(1+\epsilon)^{\beta-1}}{(\alpha(1-\alpha))^\beta} \left[\sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{t=1}^{T} v_i \rho_i \sigma_j^{\beta-1} x_{ijt}^* + \sum_{i=1}^{n} w_i \left(C_i^* - d_i\right)^+\right]. \tag{3.79}$$

Because speeds are rounded up, the completion times, and thus the tardiness can only be reduced, whereas the energy cost increases. Since at most we speed up each job by a factor $(1+\delta)$, we have that,

$$E_i(\bar{s}_i^\alpha) \leq E_i((1+\delta) s_i^\alpha) = (1+\delta)^{\beta-1} E_i(s_i^\alpha) \Rightarrow \sum_{i=1}^{n} E_i(\bar{s}_i^\alpha) \leq (1+\delta)^{\beta-1} \sum_{i=1}^{n} E_i(s_i^\alpha). \tag{3.80}$$

The approximation ratio follows from setting $\alpha = \arg\min_{0 \leq \alpha \leq 1} \left\{\frac{1}{(\alpha(1-\alpha))^\beta}\right\} = \frac{1}{2}$. We could use $\frac{(1+\epsilon)^{\beta-1}}{\alpha^\beta (1-\alpha)^{\beta-1}}$ in (3.78) to compute a tighter bound, but the resulting expression is not as clean. □

We are not able to extend this algorithm for the $1|r_i|\sum E_i(s_i) + w_iT_i$ problem, since it is based on speed scaling to make sure that jobs are finished within a desired time interval. When release dates are present, we do not see how to arbitrarily reduce the completion times.

3.3 General Energy Cost Functions

In this section we consider the extension to general energy *cost* functions, as opposed to simply energy *consumption*. We begin by considering discrete speeds, as in the previous sections, but we will relax this requirement later.

Managers of data centers are clearly interested in the energy cost metric, since they need to balance the penalty for violating the service level agreements with the cost of energy. The energy price curves for industrial consumers are often quite complicated because of energy contracts, discounts, real time pricing etc.; therefore it is very important to consider general cost functions in the scheduling model. Hence, in this section we use $\mathcal{E}_i(s_i)$ as the general energy cost function of running job i at speed s_i. We will require that $\mathcal{E}_i(s_i)$ is non-negative, just as in [Andrew *et al.*, 2009; Bansal *et al.*, 2009], but no other requirements are needed for the weighted completion time setting. For the weighted tardiness setting we will require an additional regularity condition that bounds the growth of the energy cost function.

Since in practice the processor speed can be dynamically changed during the course of a job, one can replace the general cost function by its lower convex envelope. Hence, without loss of generality, we can assume that $\mathcal{E}_i(s_i)$ is convex. Furthermore, since the machine can only run at the speeds in $\mathbf{S}$, we can also consider that $\mathcal{E}_i(s)$ is linear in between these speeds. Hence, for every $s \in [\sigma_j, \sigma_{j+1}]$ such that $s = \lambda\sigma_j + (1-\lambda)\sigma_{j+1}$, with $\lambda \in [0,1]$, then $\mathcal{E}_i(s) = \lambda\mathcal{E}_i(\sigma_j) + (1-\lambda)\mathcal{E}_i(\sigma_{j+1})$.

Note that for bounding the energy cost terms in the weighted completion time setting, we only used the fact that the energy consumption function $E_i(s) = v_i\rho_i s^{\beta-1}$ is convex. Thus, the previous bounds extend to our more general class of functions $\mathcal{E}_i(s)$. In the weighted tardiness case we also require a bound on the growth of the energy cost function, which we

will address in Section 3.3.2.

3.3.1 Weighted Completion Time Problem with General Energy Cost

The objective function (3.49) is extended as follows,

$$\min_{\mathbf{x}} \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{t=1}^{T} \left(\mathcal{E}_i(\sigma_j) + w_i \tau_{t-1}\right) x_{ijt}, \tag{3.81}$$

where $\mathcal{E}_i(\sigma_j)$ are just coefficients. Given that we only change the energy cost related terms, all the completion time related bounds computed previously are still valid.

The only modification required is in the rounding procedure at the end of the SAIAS algorithm, where it was done by rounding down the α-speeds. Now instead we will round them up or down so that $\mathcal{E}_i(\bar{s}_i^\alpha) \leq \mathcal{E}_i(s_i^\alpha)$, which is always possible since $\mathcal{E}_i(s_i)$ is linear in between the speeds in $\mathbf{S}$. With this change Lemma 3.3 remains valid and we can extend the algorithm to our general energy cost functions.

Theorem 3.8. *The SAIAS algorithm with $\alpha = \frac{1}{2}$ is a $4(1+\epsilon)(1+\delta)$-approximation algorithm for the $1|prec|\sum \mathcal{E}_i(s_i) + w_i C_i$ problem, for all general non-negative energy cost functions $\mathcal{E}_i(s)$.*

Proof. Because $\mathcal{E}_i(\sigma)$, $i = \{1, \ldots, n\}$ are convex functions, (3.25) remains valid since

$$\mathcal{E}_i(s_i^\alpha) = \mathcal{E}_i(\mathbb{E}[\hat{s}_i]) \leq \mathbb{E}[\mathcal{E}_i(\hat{s}_i)] = \sum_{j=1}^{m} \mu_{ij} \mathcal{E}_i(\sigma_j).$$

and thus, from the definition of μ_{ij}, and from $0 \leq \alpha \leq 1$, $\epsilon > 0$, and $\tilde{x}_{ijt} \leq \bar{x}_{ijt}$,

$$\sum_{i=1}^{n} \mathcal{E}_i(s_i^\alpha) \leq \frac{1}{\alpha} \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{t=1}^{I_i^\alpha} \mathcal{E}_i(\sigma_j) \tilde{x}_{ijt} \leq \frac{(1+\epsilon)}{\alpha(1-\alpha)} \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{t=1}^{T} \mathcal{E}_i(\sigma_j) \bar{x}_{ijt}. \tag{3.82}$$

The proof follows since the bounds for the completion time in Theorem 3.5 remain valid, as well as Lemma 3.3. □

By the same argument we also have that,

Theorem 3.9. *The SAIAS algorithm with* $\alpha = \sqrt{2} - 1$ *is a* $(3 + 2\sqrt{2})(1 + \epsilon)(1 + \delta)$*-approximation algorithm for the* $1|r_i, prec| \sum \mathcal{E}_i(s_i) + w_i C_i$ *problem, for all general non-negative energy cost functions* $\mathcal{E}_i(s)$*.*

3.3.2 Weighted Tardiness Problem with General Energy Cost

We replace the energy term in (3.74) with the general energy cost term to obtain the new objective

$$\min_{\mathbf{x}} \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{t=1}^{T} \left(\mathcal{E}_i(\sigma_j) + w_i \left(\tau_{t-1} - d_i \right)^+ \right) x_{ijt}. \tag{3.83}$$

Since the SAIAS-T algorithm speeds-up the jobs, we need to add the following regularity condition for the energy cost functions $\mathcal{E}_i(\sigma)$ in order to obtain performance bounds:

Assumption 3.1. $\exists \beta \in \mathbb{N}^+$*, such that* $\mathcal{E}_i(\gamma \sigma_i) \leq \gamma^{\beta-1} \mathcal{E}_i(\sigma_i), \quad \forall \gamma \geq 1.$

Theorem 3.10. *The SAIAS-T algorithm with* $\gamma = \frac{(1+\epsilon)}{\alpha(1-\alpha)}$ *and* $\alpha = \frac{1}{2}$*, is a* $4^{\beta}(1 + \epsilon)^{\beta-1}(1 + \delta)^{\beta-1}$*-approximation algorithm for the* $1|prec| \sum \mathcal{E}_i(s_i) + w_i T_i$ *problem, for all non-negative energy cost functions* $\mathcal{E}_i(s)$ *that satisfy Assumption 3.1.*

Proof. As before, all the completion time related bounds (3.76) and (3.77) remain valid, so only a bound analogous to (3.78) is needed. From Assumption 3.1 it follows that,

$$\mathcal{E}_i(\tilde{s}_i^{\alpha}) \leq \gamma^{\beta-1} \mathcal{E}_i(s_i^{\alpha}) \leq \frac{(1+\epsilon)^{\beta-1}}{\alpha^{\beta}(1-\alpha)^{\beta}} \sum_{j=1}^{m} \sum_{t=1}^{T} \mathcal{E}_i(\sigma_j) \bar{x}_{ijt}. \tag{3.84}$$

Thus, from (3.77) it follows that,

$$\sum_{i=1}^{n} \mathcal{E}_i(\tilde{s}_i^{\alpha}) + \sum_{i=1}^{n} w_i \left(\tilde{C}_i^{\alpha} - d_i \right)^+ \leq \frac{(1+\epsilon)^{\beta-1}}{\alpha^{\beta}(1-\alpha)^{\beta}} \left[\sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{t=1}^{T} \mathcal{E}_i(\sigma_j) x_{ijt}^* + \sum_{i=1}^{n} w_i \left(C_i^* - d_i \right)^+ \right]. \tag{3.85}$$

Since we are rounding speeds up, equation (3.80) remains valid and thus taking $\alpha = \frac{1}{2}$ completes the proof. □

3.3.3 Continuous Speeds

As commented previously, our algorithms are also applicable for the case when a continuous set of speeds is possible. In this case we modify the SAIAS and SAIAS-T algorithms, eliminating the rounding step required at the end of each algorithm.

When the operating range of the machine is given, i.e. the speed limits $\sigma_{\min}$ and $\sigma_{\max}$, since our IP requires a speed index, we need to quantize the set $[\sigma_{\min}, \sigma_{\max}]$ in m different speeds. We can do this by setting $\sigma_1 = \sigma_{\min}$, and as before we define speed $\sigma_j = (1+\delta)\sigma_{j-1}$, for some $\delta > 0$, making sure that $\sigma_m \geq \sigma_{\max}$ in order to cover the whole operating range. Just by rounding as described in Section 3.3.1 for the weighted completion time setting and rounding up for the weighted tardiness setting we can prove the following lemma:

Lemma 3.7. *The optimal solution for the IP (3.49)-(3.54) is at most* $(1+\delta)$ *times the optimal solution of the energy aware problem in the weighted completion time and continuous speed setting, and the optimal solution for the IP (3.74), (3.50)-(3.54) is at most* $(1+\delta)^{\beta-1}$ *times the optimal solution of the energy aware problem in the weighted tardiness and continuous speed setting.*

The proof is analogous to Lemma 3.3 for the weighted completion time and similar to equation (3.80) for the weighted tardiness setting. Since there is no additional rounding at the end of the algorithm, using Lemma 3.7 we get the same approximation ratios as in Theorems 3.8, 3.9, and 3.10.

When the operating range of the machine is not given, and we are interested in determining a set $\mathbf{S}$ that covers the optimal speeds from the continuous case, we need the following additional regularity condition on the energy cost functions: $\exists \xi < \infty$ such that $\mathcal{E}_j(s_i)$ is increasing $\forall s_i \geq \xi$. It is easy to prove that this is a necessary and sufficient conditions for the problem to be well defined, and thus we can compute $\sigma_{\min}$ and $\sigma_{\max}$ such that the optimal speeds $s_i^* \in [\sigma_{\min}, \sigma_{\max}]$, for all i. Then we can apply the same procedure as before to quantize and build the set of speeds.

Chapter 4

Heuristics and Experimental Results for EAS

THROUGHOUT this chapter we present several heuristics that improve the performance or extend the applicability of our algorithms to other settings not considered in our theoretical results. We also present several experimental results that show that the performance of our algorithms is much better than our theoretical worst-case bounds and analyze the magnitude of the heuristic improvements. Additionally, we present the performance when other settings are considered such as weighted flow time as scheduling metric, or when multiple machines are available.

4.1 Heuristic Improvement for Weighted Completion Time

A natural improvement for the SAIAS algorithm is to recalculate the optimal speeds once the order is defined by the algorithm. Without loss of generality, we assume that the schedule order computed by the SAIAS algorithm is $\Pi^{\alpha} = \{1, 2, \ldots, n\}$. The following result establishes that we can compute the optimal processing speed for each job, for any given order, in closed form. This result is a detailed extension of the results given in Lemma 3.1.

Lemma 4.1. *Given the schedule order Π^α, the optimal speed at which to run job i is given by*

$$s_i^* = \sqrt[\beta]{\frac{\sum_{j=i}^{n}\sum_{k=1}^{i}\lambda_{jk}^*}{(\beta-1)v_i}},\ \forall i \in \{1,\dots,n\}\ , \tag{4.1}$$

where λ_{jk}^* is the optimal solution of the following optimization problem:

$$\begin{aligned} \max_{\boldsymbol{\lambda}}\ & \sum_{i=1}^{n}\sum_{j=1}^{i}\lambda_{ij}r_j + \sum_{i=1}^{n}\mathcal{B}\rho_i v_i^{\frac{1}{\beta}}\left(\sum_{j=i}^{n}\sum_{k=1}^{i}\lambda_{jk}\right)^b \\ s.t.:\ & \sum_{j=1}^{i}\lambda_{ij} = w_i, \quad \forall i \\ & \lambda_{ij} \geq 0, \quad \forall j \in \{1,\dots,i\}, \forall i, \end{aligned} \tag{4.2}$$

with $\mathcal{B} \equiv \frac{\beta}{(\beta-1)^b}$ and $b \equiv \frac{\beta-1}{\beta}$.

Proof. Given the order $\Pi^\alpha = \{1,2,\dots,n\}$ the optimal speeds are given by the solution of the following optimization problem:

$$\begin{aligned} \min_{\mathbf{s}}\ & \sum_{i=1}^{n} v_i\rho_i s_i^{\beta-1} + w_i C_i \\ \text{s.t.:}\ & C_i \geq r_j + \sum_{k=j}^{i}\frac{\rho_k}{s_k}, \quad \forall j \in \{1,\dots,i\},\ \forall i \\ & s_i \geq 0, \quad \forall i. \end{aligned} \tag{4.3}$$

The Lagrangian for (4.3) is given by

$$\mathcal{L}(\mathbf{s},\boldsymbol{\lambda}) = \sum_{i=1}^{n} v_i\rho_i s_i^{\beta-1} + w_i C_i - \sum_{i=1}^{n}\sum_{j=1}^{i}\lambda_{ij}\left(C_i - r_j - \sum_{k=j}^{i}\frac{\rho_k}{s_k}\right)\ , \tag{4.4}$$

where λ_{ij}, $j = \{1,\dots,i\}$ denotes the dual variables of the release date constraints in (4.3) for the i-th job. From the necessary conditions for optimality it follows that $\sum_{j=1}^{i}\lambda_{ij}^* = w_i$, for all $i \in \{1,\dots,n\}$, and that the optimal speed for job i is given by (4.1), where $\mathbf{s}^*$ and $\boldsymbol{\lambda}^*$ are the optimal speeds and optimal dual variables respectively.

Using (4.4) it is also easy to show that (4.2) is the dual problem of (4.3). Since $\boldsymbol{\lambda}^*$ is its optimal solution, it follows that (4.1) will give the optimal speeds. □

Note that from (4.1) we get that the optimal speed of the i-th job only depends of the dual variables of the completion time constraints of future jobs, and not past ones.

Corollary 4.1. *If $r_i = 0$, $\forall i$, i.e. all jobs are available at time 0, then the optimal speed of job i is given by*

$$s_i^* = \sqrt[\beta]{\frac{\sum_{j=i}^{n} w_j}{(\beta-1)v_i}}, \ \forall i \in \{1, \ldots, n\} \ . \tag{4.5}$$

Proof. By setting $r_i = 0$, $\forall i$ in (4.2) we note that the maximum value of this modified optimization problem is achieved when $\lambda_{i1} = w_i$, and thus $\lambda_{ij} = 0$, for $j = \{2, \ldots, i\}$. The proof follows by using these values of $\boldsymbol{\lambda}$ in (4.1). □

This result is an extension of the speed rule used in most of the energy aware scheduling literature for the flow time metric [Andrew *et al.*, 2010; Andrew *et al.*, 2009]. The main result in these papers is that using SRPT as ordering policy and a speed of $s = \sqrt[\beta]{\frac{n_t}{(\beta-1)v}}$, where n_t is the number of jobs available at time t, achieves the best known competitive ratios. Since [Andrew *et al.*, 2010; Andrew *et al.*, 2009] consider the total flow time (i.e. $w_i = 1$, $\forall i$) and the same energy cost function for all jobs (i.e. $v_i = v$, $\forall i$) the optimal speed $s = \sqrt[\beta]{\frac{n_t}{(\beta-1)v}}$ is identical to the one given in (4.5).

Using Lemma 4.1 and Corollary 4.1 one can design an algorithm that computes the optimal speeds for a given order Π in $O(n)$ time, when there are no release dates, and in $O(n^2)$ time, when there are release dates. We describe these algorithms ub the following section, since this setting is a special case of the more general setting detailed in Section 4.2.

4.2 Heuristic Improvement for Weighted Tardiness

The total weighed completion time is a special case of the total weighted tardiness, in which all deadlines are set to $d_i = 0$. In this section we explore the weighted tardiness setting, describing a similar heuristic improvement as the one detailed in Section 4.1 and a simple

polynomial time algorithm that can compute the optimal speeds without solving the nonlinear optimization problem.

4.2.1 Optimality Conditions for Speed-Scaling

W.l.o.g., we can assume that $\Pi^\alpha = \{1, 2, \ldots, n\}$. Then, problem (3.73) can be reformulated as:

$$\min_{\mathbf{s},\mathbf{z}} \sum_{i=1}^{n} v_i \rho_i s_i^{\beta-1} + w_i z_i \tag{4.6}$$

$$\begin{aligned} \text{s.t.:} \quad & \\ z_i &\geq \textstyle\sum_{j=1}^{i} \frac{\rho_j}{s_j} - d_i, \quad \forall i \in \{1, \ldots, n\} \\ z_i &\geq 0, \qquad\qquad\quad \forall i \in \{1, \ldots, n\} \\ \mathbf{s} &\geq 0. \end{aligned}$$

Using standard Lagrangian theory, we can look for the necessary optimality conditions, with

$$\mathcal{L}(\mathbf{s}, \mathbf{z}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\eta}) = \sum_{i=1}^{n} v_i \rho_i s_i^{\beta-1} + w_i z_i - \lambda_i \left(z_i - \sum_{j=1}^{i} \frac{\rho_j}{s_j} + d_i \right) - \mu_i z_i - \eta_i s_i. \tag{4.7}$$

The optimality conditions state that if $(\mathbf{s}, \mathbf{z})$ is a local minimum and is regular, and the constraints are differentiable, then there exists Lagrange multipliers $\lambda \geq 0$, $\mu \geq 0$, and $\eta \geq 0$ such that:

$$\frac{\partial \mathcal{L}}{s_l} = (\beta - 1) v_l \rho_l s_l^{\beta-2} - \frac{\rho_l}{s_l^2} \sum_{j=l}^{n} \lambda_j - \eta_l = 0, \ \forall l \tag{4.8}$$

$$\frac{\partial \mathcal{L}}{z_l} = w_l - \lambda_l - \mu_l = 0, \ \forall l. \tag{4.9}$$

Now, by complementary slackness we have that $\eta_i s_i = 0, \ \forall i$. Since $s_i > 0, \ \forall i$ (otherwise the cost function will go to $+\infty$), then it must be that $\eta_i = 0, \ \forall i$.

We can classify jobs into 3 different cases: job i is early if $c_i < d_i$, is on time if $c_i = d_i$, or is late if $c_i > d_i$.

According to this classification, if a job is late then $c_i - d_i > 0$. Since $z_i \geq c_i - d_i$ then $z_i > 0$. By complementary slackness $\mu_i z_i = 0, \ \forall i$ which implies that if a job l is late

then $\mu_l = 0$, and thus $\lambda_l = w_l$. On the other hand, if a job l is early then $c_i - d_i < 0$, making $z_l = 0$ and thus $z_l - c_l + d_l \neq 0$. Again, by complementary slackness, we have that $\lambda_i(z_i - c_i + d_i) = 0,\ \forall i$, so if job l is early $\lambda_l = 0$ and $\mu_l = w_l$. Finally, if job l is on time $c_l - d_l = 0$ and $z_l = 0$. As $\mu_l \geq 0$ we will then have that $w_l \geq \lambda_l \geq 0$.

By equation 4.8 we also get that, since $\eta_l = 0$, then

$$s_l = \left(\frac{\sum_{j=l}^{n} \lambda_j}{(\beta - 1)v_l} \right)^{\frac{1}{\beta}}. \tag{4.10}$$

Equation 4.10 indicates that if future jobs are finishing early, we can select a slower speed for the current job in order to reduce its cost and at the same time push these future jobs further near their deadlines. On the contrary, if all future jobs are late, the speed of the current job will be high and thus future jobs end earlier. An additional insight we get from equation 4.10 is the following Lemma.

Lemma 4.2. *In the EAS problem with total weighted tardiness as scheduling metric, given any order* Π*, the* n^{th} *job cannot finish early in the optimal solution for the speeds, i.e.* $C_n \geq d_n$.

Proof. We prove this by contradiction. If job n is early, then by equation (4.10) we have that $s_n = 0$, since $\lambda_n = 0$, and we this will make the cost go to $+\infty$. Hence, this cannot be an optimal solution. □

This is similar to the result we obtained in Lemma 3.6 as it basically implies that there is no idle time at the end either.

Using the Lagrangian from equation (4.7) we can also derive a dual of Problem (4.6). Rearranging the terms in equation (4.7) we get that the dual problem is determined by

$$\max_{\boldsymbol{\lambda},\boldsymbol{\mu},\boldsymbol{\eta} \geq 0} \left\{ \min_{\mathbf{s},\mathbf{z}} \left\{ \sum_{i=1}^{n} v_i \rho_i s_i^{\beta-1} + z_i \left(w_i - \lambda_i - \mu_i \right) - \eta_i s_i - \lambda_i d_i + \sum_{j=1}^{i} \lambda_i \frac{\rho_j}{s_j} \right\} \right\}. \tag{4.11}$$

We need to have that $w_i - \lambda_i - \mu_i = 0$ for this problem to be well defined, as well as $\eta_i = 0,\ \forall i$. Furthermore, since $\mu \geq 0$ we rewrite the constraints as $0 \leq \lambda \leq w$. Then, we have that

$$\max_{0 \leq \boldsymbol{\lambda} \leq w} \left\{ \sum_{i=1}^{n} -\lambda_i d_i + \min_{\mathbf{s}} \left\{ \sum_{i=1}^{n} v_i \rho_i s_i^{\beta-1} + \sum_{i=1}^{n} \sum_{j=1}^{i} \lambda_i \frac{\rho_j}{s_j} \right\} \right\}. \tag{4.12}$$

We now exchange the sums in the second term of the minimization problem in equation (4.12), and we get that

$$\begin{aligned} F_{\boldsymbol{\lambda}}(\mathbf{s}) &= \sum_{i=1}^{n} v_i \rho_i s_i^{\beta-1} + \sum_{i=1}^{n}\sum_{j=1}^{i} \lambda_i \frac{\rho_j}{s_j} = \sum_{i=1}^{n} v_i \rho_i s_i^{\beta-1} + \sum_{i=1}^{n}\sum_{j=i}^{n} \lambda_j \frac{\rho_i}{s_i} \\ &= \sum_{i=1}^{n} \left(v_i \rho_i s_i^{\beta-1} + \Lambda_i \frac{\rho_i}{s_i} \right), \end{aligned} \tag{4.13}$$

where $\Lambda_i \equiv \sum_{j=i}^{n} \lambda_j$. With this transformation the minimization problem becomes an unconstrained and decoupled minimization problem with the following optimal solution:

$$\frac{\partial F_{\boldsymbol{\lambda}}(\mathbf{s})}{s_i} = (\beta-1) v_i \rho_i s_i^{\beta-2} - \frac{\rho_i}{s_i^2}\Lambda_i = 0 \Rightarrow s_i^* = \left(\frac{\sum_{j=i}^{n} \lambda_j}{(\beta-1)v_i} \right)^{\frac{1}{\beta}}, \ \forall i. \tag{4.14}$$

Hence, the optimal value is

$$F_{\boldsymbol{\lambda}}(\mathbf{s}^*) = \sum_{i=1}^{n} \mathcal{B} \rho_i v_i^{\frac{1}{\beta}} \left(\sum_{j=i}^{n} \lambda_j \right)^{b}, \tag{4.15}$$

where $\mathcal{B} \equiv \frac{\beta}{(\beta-1)^b}$ and $b \equiv \frac{\beta-1}{\beta}$.

Evaluating solution (4.15) in (4.12) we get that the dual problem is

$$\max_{\boldsymbol{\lambda}} \sum_{i=1}^{n} -\lambda_i d_i + \mathcal{B} \rho_i v_i^{\frac{1}{\beta}} \left(\sum_{j=i}^{n} \lambda_j \right)^{b} \tag{4.16}$$

s.t.:

$$0 \leq \lambda_i \leq w_i, \quad \forall i \in \{1, \ldots, n\}.$$

Considering that no order is given, then the dual of the combinatorial problem (3.73) becomes:

$$\min_{\Pi} \ \max_{\lambda} \sum_{i=1}^{n} -\lambda_i d_{\pi(i)} + \mathcal{B} \rho_{\pi(i)} v_{\pi(i)}^{\frac{1}{\beta}} \left(\sum_{j=i}^{n} \lambda_{\pi(j)} \right)^{b} \tag{4.17}$$

s.t.:

$$0 \leq \lambda_i \leq w_i, \quad \forall i \in \{1, \ldots, n\}.$$

Although we can solve problem (4.16) easily, since it is a convex optimization problem, there is a special case where we can compute the optimal speeds without solving the nonlinear optimization problem. This is when all jobs have the same deadline. A special case of this setting is the total weighted completion time, since in that setting all jobs have the deadline $d_i = 0$.

4.2.2 Special Case: Common Deadline

When all jobs have a common deadline (i.e. $d_i = d,\ \forall i$), given an order Π, we know that if job m is on time (or in absence of an on-time job, m is the first late job), all following jobs will be late, and all the previous ones will be early. This simplifies the problem since now we only need to find which job is this first on-time or late job to compute the optimal speeds. If we know that job, then we assign $\lambda_i = 0,\ \forall i < m$ (i.e. all the previous jobs), and $\lambda_i = w_i,\ \forall i > m$ (i.e. for the jobs after m). Finally, if job m is late we assign $\lambda_m = w_m$, and if it is on time we use the following lemma.

Lemma 4.3. *In the setting where $d_i = d \geq 0$ for all i, if a job m is completed on-time, its Lagrange multiplier λ_m has the value*

$$\lambda_m = \frac{\beta - 1}{d^\beta} \left(\sum_{i=1}^{m} \rho_i v_i^{\frac{1}{\beta}} \right)^\beta - \sum_{i=m+1}^{n} w_i. \tag{4.18}$$

Proof. Since all the jobs after m are late, $\lambda_i = w_i,\ \forall i > m$. Then, using equation (4.10) we have that

$$s_m = \left(\frac{\sum_{j=m}^{n} \lambda_j}{(\beta - 1) v_m} \right)^{\frac{1}{\beta}} = \left(\frac{\lambda_m + \sum_{j=m+1}^{n} w_j}{(\beta - 1) v_m} \right)^{\frac{1}{\beta}}.$$

Now for jobs that finish earlier than m, all of which are early jobs, we have that $\lambda_i = 0,\ \forall i < m$. Hence, the speed at which these jobs run is

$$s_i = \left(\frac{\sum_{j=i}^{n} \lambda_j}{(\beta - 1) v_i} \right)^{\frac{1}{\beta}} = \left(\frac{\lambda_m + \sum_{j=m+1}^{n} w_j}{(\beta - 1) v_i} \right)^{\frac{1}{\beta}}.$$

On the other hand, since job m finishes on time we know that $C_m = d$, and thus

$$C_m = \sum_{i=1}^{m} \frac{\rho_i}{s_i} = \sum_{i=1}^{m} \rho_i \left(\frac{(\beta - 1) v_i}{\lambda_m + \sum_{j=m+1}^{n} w_j} \right)^{\frac{1}{\beta}} = \frac{(\beta - 1)^{\frac{1}{\beta}}}{\left(\lambda_m + \sum_{j=m+1}^{n} w_j \right)^{\frac{1}{\beta}}} \sum_{i=1}^{m} \rho_i v_i^{\frac{1}{\beta}} = d.$$

Solving the previous equation for λ_m proves the Lemma. □

Using Lemma 4.3 and equation (4.10) we can design two algorithms to compute the optimal speeds of an instance given an order Π.

Algorithm 4.1 PRIMAL SPEED

Inputs: set of n jobs and order Π.

Set $\lambda_i = 0,\ \forall i$, and compute speeds s_i and completion time C_i.
Set $i = n$ and $d_t = \infty$.
while $d_t > d$
    if $C_i < d$
        Stop.
    elseif $C_i \geq d$
        Assume i is late.
        Set $\lambda_i = w_i$ and then calculate s_j and C_j, $\forall j$.
        if $c_i \geq d$
            Keep job i late.
            Set $i = i - 1$ and $d_t = c_i$.
        elseif $C_i < d$
            Job i should finish at time d.
            Set $\lambda_i = \frac{\beta-1}{d^\beta} \cdot \left(\sum_{j=1}^{i} \rho_j v_j^{\frac{1}{\beta}} \right)^\beta - \sum_{j=i+1}^{n} w_j$.
Compute s_i and C_i, $\forall i$.
return speeds **s**.

4.2.2.1 Primal Speed Algorithm

First we use the prime problem formulation (4.6) to design an algorithm that will compute the optimal speeds. We name this algorithm PRIMAL SPEED and we detail in Algorithm 4.1.

The PRIMAL SPEED Algorithm works as follows: we start from the last job, job n, and check if this should be an early or a late job. Then we compute the speeds according to the assumption and check if the finishing times actually match our assumption. The main idea is that if we have already defined that jobs $p+1$ to n are late, we can test if job p is late or not by changing its value of λ_p and then checking what happens with C_p. If $C_p < d$ from the

start, we stop since if we increase λ_p we will only make p and all the previous jobs faster, hence we will keep having $C_p < d$. If this is not the case and $C_p \geq d$, we assume that p must then be late: $\lambda_p = w_p$. That makes the speed of all previous jobs higher than before and reduces C_p. If after that we still have that $C_p > d$ then we were right with our assumption and job p will always be late. On the contrary if now that the speeds are higher we have that $C_p < d$, we have a contradiction, since this means that job p finishes early. Hence, we have shown that keeping $\lambda_p = 0$ makes the job late, but setting it to $\lambda_p = w_p$ will make it early, exactly the opposite of what we saw were the optimality conditions. Thus, λ_p must be in between these two values, which only occurs if a job is on time, and we can use Lemma 4.3 to compute the exact value of λ_p.

Theorem 4.1. *The* Primal Speed *algorithm returns the optimal speeds given an order* Π *after* $O(n^3)$ *operations.*

Proof. The correctness of the algorithm is easy to prove by checking the optimality conditions of the algorithm's output. Assuming that the algorithm labels correctly which jobs are late and which early, the output complies with the KKT conditions, as all jobs labeled late will have $\lambda_i = w_i$, all jobs labeled early will have $\lambda_i = 0$, and the speed is determined according to equation 4.10. Additionally, the stopping condition makes sure that all jobs labeled late finish after d and all jobs labeled early finish before. By contradiction, we can assume the algorithm finishes with a solution $\tilde{s}$ that is not the optimal one s^*. We have two cases: one in which there is a job i that is on-time (with $0 < \tilde{\lambda}_i < w_i$) and one in which there is no job satisfying this.

If job i is on-time, as the definition of which jobs are early and which late determines completely the speeds of all the jobs, this means that in s^* this job must be either early or late. If job i is early in s^*, (thus $\lambda_i^* = 0 < \tilde{\lambda}_i$) then all $s_j^* < \tilde{s}_j$ for $j \geq i$ and the completion time of job i will be later: $C_i^* > \tilde{C}_i$. That is a contradiction since $\tilde{C}_i = d$ and thus in s^* job i must then be late. Now if we consider that job i is late in s^* then $\lambda_i^* = w_i > \tilde{\lambda}_i$, making all previous jobs run faster: $s_j^* > \tilde{s}_j$. This implies that the completion time of i in the optimal case must be earlier than for $\tilde{s}$, $C_i^* < \tilde{C}_i$. Again this is a contradiction since that means that

in the optimal case job i must be early.

The case where there is no on-time job is analogous but with more cases, as now we can show that neither the first late job of the algorithm's solution can be early or on-time in the optimal one, nor the last early can be late or on-time in the optimal solution, which again means that the optimal solution and the algorithm's output must be the same.

It is easy to show that the algorithm will finish since once we decide that a job is late, it will never become early again in future iterations, because of the way in which the decision of a job being late or not is taken. As the speed of a job depends on the values of λ for all the following jobs, the only way a job i, already labeled as late, could become early is that all previous jobs run at a faster speed, bringing its completion time before d. But that means that some λ_j, with j being an early job, must increase which is a contradiction since that implies job j must become late or on time, and thus i will still be late or on time. Hence, once a job is labeled as late, it will be always late, and as we have a finite number of jobs the algorithm will finish.

In terms of complexity, the initialization requires $O(1)$ operations to be performed. Next, as there exists only n jobs and we will never check one twice, after $O(n)$ iterations the algorithm will stop. Furthermore, on each iteration we need to calculate $O(n)$ speed values and $O(n)$ completion times, each requiring $O(n)$ operations. Hence we will obtain the optimal solution in $O(n^3)$ operations. □

4.2.2.2 Dual Speed Algorithm

Although the Primal Speed algorithm is $O(n^3)$, we can do better by using the dual formulation (4.17). We call this algorithm Dual Speed and it is detailed in Algorithm 4.2.

Theorem 4.2. *The* Dual Speed *algorithm computes the optimal speed values given an order* Π *after* $O(n)$ *operations.*

Proof. The correctness of the algorithm is analogous to the proof in Theorem 4.1, since the optimality conditions are attained at the end of the algorithm.

Algorithm 4.2 DUAL SPEED

Inputs: set of n jobs and order Π.

Set $V = \sum_{j=1}^{n} \rho_j v_j^{\frac{1}{\beta}}$, $W = 0$, and $\lambda_j = 0,\ \forall j$.
Set $i = n$ and $B \equiv \frac{\beta-1}{d^\beta}$.
while $i > 0$
    Set $\lambda_i = BV^\beta - W$.
    if $0 \leq \lambda_i < w_i$
        Stop.
    elseif $\lambda_i < 0$
        Set $\lambda_i = 0$ and **Stop**.
    elseif $\lambda_i \geq w_i$
        Set $\lambda_i = w_i$.
        Update $V = V - \rho_i v_i^{\frac{1}{\beta}}$ and $W = W + w_i$.
        i = i - 1.
Compute all speeds s_i.
return speeds **s**.

As in Theorem 4.1, because there exist only n different λ_i and we will never set the value of one twice, after $O(n)$ iterations the algorithm will stop. Furthermore, on each iteration we will do at most $O(1)$ operations, just set the value of λ_i, and update i, V, and W. Since in this case the initialization requires $O(n)$ operations to get the initial value of V, after $O(n)$ operations the algorithm will compute the optimal solution. □

This algorithm significantly reduces the operations required to compute the optimal speeds and can be directly applied to the total weighted completion time by setting $d = 0$.

4.2.2.3 Cases Solvable In Polynomial Time

To conclude this special case we observed though simulations that Theorem 3.1 also produced the optimal schedule in this setting, without any counterexamples. Regretfully, the same

proof does not work in this case so we leave it as an open conjecture.

Conjecture 4.1. *For the case where all jobs have the same deadline $d_i = d > 0$, if $w_i = w,\ \forall i$ or $\rho_i v_i^{\frac{1}{\beta}} = \xi,\ \forall i$ then the order Π is optimal if*

$$\frac{w_{\pi(i)}}{\rho_{\pi(i)} v_{\pi(i)}^{\frac{1}{\beta}}} \geq \frac{w_{\pi(i+1)}}{\rho_{\pi(i+1)} v_{\pi(i+1)}^{\frac{1}{\beta}}}.$$

4.3 Experimental Results

The theoretical bounds in Chapter 3 do not appear to be tight. In order to better understand the performance of our algorithms, we analyze them experimentally. The main contribution of this Section is twofold. First we present the experimental performance analysis of our algorithms and show that although the theoretical bounds are relatively small, in practice the bounds are even better and the algorithms perform very close to optimal. Additionally, we also extend the algorithms to more complex settings: online problems, multiple machines, and using total weighted flow time as the scheduling metric, showing good empirical approximation ratios as well. Table 4.1 shows a summary of the main results included in this chapter.

It is important to note that there are very few examples of performance analysis in the energy aware scheduling literature with a notable exception being [Andrew *et al.*, 2010].

4.3.1 Experimental Performance for Weighted Completion Time

In this section we present a simulation based analysis of the performance of the SAIAS algorithm for the total weighted completion time setting.

For each analysis we simulated a large number of randomly generated instances with the following distributions: $v_i \sim \text{unif}\{0, \ldots, 40\}$, $w_i \sim \text{unif}\{0, \ldots, 20\}$, and $\rho_i \sim \text{unif}\{1, \ldots, 10\}$. Although the size of the jobs seems small, we also analyzed instances with much larger ones (such as $\rho_i \sim \text{unif}\{1, \ldots, 100\}$), as well as ρ_i drawn from bimodal distributions, which are

Problem	Algorithm	Average Ratio	Worst Ratio	Equal Instances
$1\|prec\|\sum E_i(s_i) + w_iC_i$	SAIAS	1.0077	1.1437	28.39%
	SAIAS-H	1.0045	1.0997	48.07%
$1\|r_i, prec\|\sum E_i(s_i) + w_iC_i$	SAIAS	1.0328	1.5133	1.34%
	SAIAS-H	1.0128	1.3025	40.06%
Polynomial Energy Cost Function	SAIAS	1.0360	1.9785	40.45%
Online Setting	SAIAS-H Online	1.1121	2.4715	0.69%
$P\|\|\sum E_i(s_i) + w_iC_i$	SAIAS-P	1.0810	1.2475	0%
$1\|prec\|\sum E_i(s_i) + w_iT_i$	SAIAS-T	3.5464	22.7207	0%
	SAIAS-T NS	1.2645	2.4645	0.49%
$1\|r_i, prec\|\sum E_i(s_i) + w_iT_i$	SAIAS-T	2.8929	16.2307	0%
	SAIAS-T NS	1.2774	2.1789	0.16%

Table 4.1: Experimental Results Summary

generally hard for scheduling algorithms, without observing any significant degradation in the performance of our algorithms.

We compared the output of our algorithm with the integer solution of the interval-and-speed-indexed formulation (IPi), its linear relaxation (LPi), and the integer and relaxed solutions of a time-and-speed-indexed formulation for this problem (IPt and LPt respectively). All simulations were done in Matlab, using Gurobi [Gurobi Optimization Inc., 2012] and Gurobi MEX [Yin, 2012] to solve the IP and LP relaxations of each instance.

For each setting, we also applied the heuristic improvement discussed in Section 4.1. We denoted this algorithm as SAIAS-H which is detailed in Algorithm 4.3.

Our experimental results show that the SAIAS algorithm, although it has a theoretical

Algorithm 4.3 SAIAS-H

1 Apply the SAIAS algorithm to the problem to compute $\mathbf{s}^\alpha$, Π^α, and $\mathbf{C}^\alpha$.

2 Compute the optimal speed s_i^* each job i, given the schedule Π^α.

3 Round each speed s_i^* to $\bar{s}_i$, the closest speed in $\mathbf{S}$, and calculate new completion times $\bar{\mathbf{C}}$.

4 **return** the cheapest solution between $\mathbf{s}^\alpha$, Π^α, and $\mathbf{C}^\alpha$ or $\bar{\mathbf{s}}$, Π^α, and $\bar{\mathbf{C}}$.

Problem	Instances	Size (n)	Average Ratio	99.5%	Worst Ratio
$1\|r_i, prec\|\sum E_i(s_i) + w_i C_i$	20,000	7	1.055	1.231	1.420
	20,000	100	1.135*	1.218	1.273
	20,000	500	1.133*	1.157	1.184
	3,000	1,000	1.136*	1.150	1.155
SAIAS-H	20,000	7	0.991	1.000	1.000
	20,000	100	0.991	0.994	0.995
Online (no *prec*)	20,000	7	1.141	1.600	2.456
	20,000	100	1.397*	1.496	1.627

Table 4.2: Experimental Results Summary for Total Weighted Completion Time

approximation ratio of $3+2\sqrt{2}+\epsilon \approx 5.8+\epsilon$ for the case with arbitrary precedence constraints and arbitrary release dates, in practice it performs very close to optimal, with average approximation ratios below 1.14. Furthermore we also show that these results remain stable even when the size of the instances grow several orders of magnitude. It is important to note that when analyzing large instances, since the IPt formulation is too large to be solved in a reasonable time, we compared the algorithm's output with the LPi solution, and thus the real approximation ratio is likely to be even better. The results also show that the SAIAS-H algorithm, where we compute the optimal speeds given the order computed by the SAIAS algorithm, reduces the approximation ratios even further. This improvement can also be used in the online setting.

In the following subsections we present details of the simulation results. We characterize the distribution of the approximation or competitive ratios via histograms. We believe that displaying the entire distribution is important since it gives a more complete understanding of how the algorithm performs as compared to just reporting an average value or a worst case scenario. In the histograms we highlight the average value for all simulations and the 99.5% quantile. For both these measures we also display the 99.99% confidence intervals, shown as doted lines around the corresponding value. Table 4.2 shows a summary of the

main experimental results for the $1|r_i, prec|\sum E_i(s_i) + w_i C_i$ problem.

4.3.1.1 SAIAS Performance

The number of variables and constraints in the IPt formulation grows very fast with the number of jobs n, making it impractical for large instances. On the other hand, the size of the IPi formulation can be easily controlled with ϵ and thus much larger instances can be simulated. For small instances we compared the performance of the SAIAS algorithm to the optimal solution of IPt, as well as the bounds given by the LPi formulation. For large instances we compared the SAIAS algorithm's solution to the LPi bound.

The simulation settings for the smaller instances were: $20,000$ simulation with $n = 7$ jobs, $\alpha = \sqrt{2} - 1$, $\delta = 0.5$, $\epsilon = 0.1$, and $r_i \sim \text{unif}[0, 0.1 \sum_i \frac{\rho_i}{\sigma_1}]$. The upper bound of the

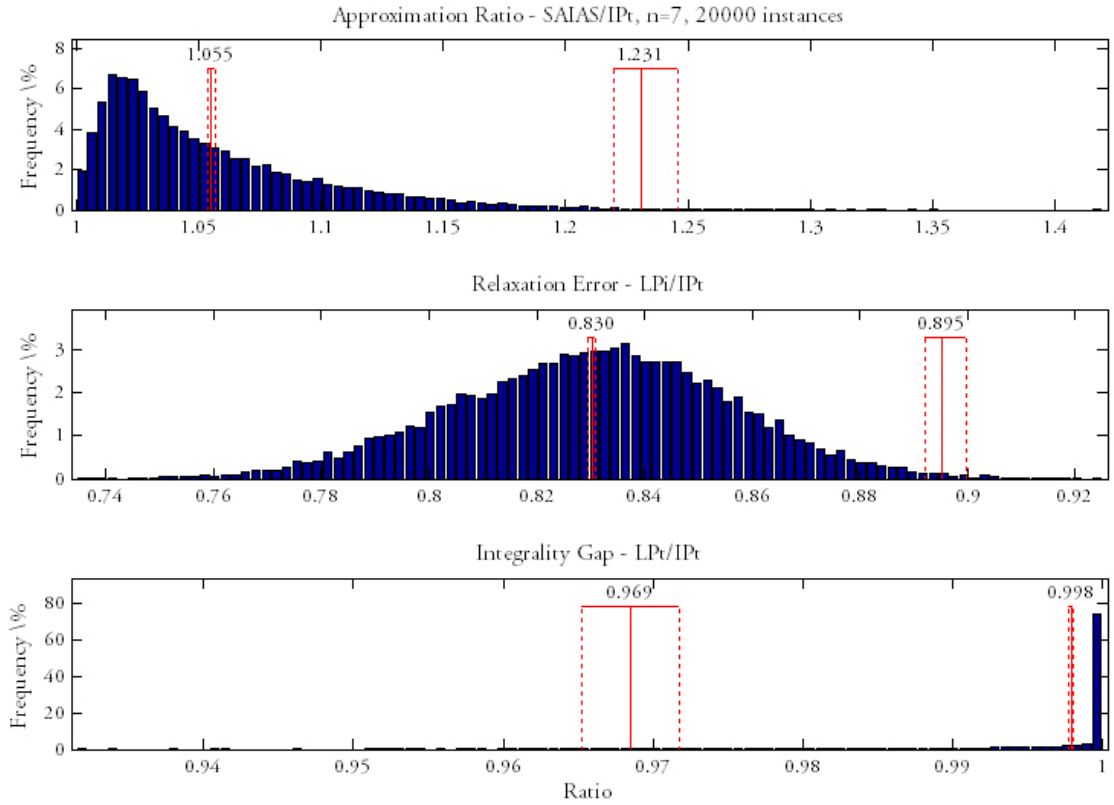


Figure 4.1: Ratios for Total Weighted Completion Time, with $n = 7$.

probability distribution of the release dates was determined experimentally to show the worst performance, particularly in the on-line settings we present later. The reason is that a smaller upper bound will be equivalent to have an offline setting, whereas a very large upper bound makes most jobs to be separate from others, and thus we don't have a set of jobs to order. Figure 4.1 shows the main results of the simulations.

The first histogram displays that the empirical approximation ratio of the SAIAS algorithm is close to optimal with an average of 1.055, and the 99.5% quantile given by 1.231. For the $20,000$ instances, the worst approximation ratio was 1.420.

The second histogram displays the difference between the LPi bound and the IPt optimal solutions. This is important since later we will compare the output of our algorithm to the LPi solution. The second histogram shows that on average the LPi bound is 0.830 times the IPt solution, and in 99.5% of the instances it was below 0.895. This means that when we compare the algorithm's output to the LPi we need to remember that we are comparing it with a solution that is between 10% to 20% lower than the integer optimal solution.

The last histogram shows that the integrality of the IPt formulation gap is very small, with an average value of 0.998 and 99.5% of the instances resulted in relaxation values above 0.969 times the value of the IPt solution.

Since our experiments show a stable and relatively constant ratio between IPt and LPi, for larger instances we compare the performance of the SAIAS algorithm to the LPi bound. Given that we don't solve the IPt formulation the simulations can be done over much larger instances. Figure 4.2 shows the result of $n = 500$ (with $20,000$ random instances) and $n = 1,000$ (with $3,000$ random instances) jobs.

Although we are now comparing the algorithm's output to the LPi solution, the approximation ratio remains small. For $n = 500$ we have an average ratio of 1.133, a worst value of 1.184, and 99.5% of the cases below 1.157; and an average ratio of 1.136, a worst value of 1.155, and 99.5% of the cases below 1.150 when $n = 1,000$. The fact that the approximation ratio is not much bigger is important since for $n = 7$ the ratio between the LPi and the IPt solution was in average 0.83, hence much of the error shown in figure 4.2 could be attributed

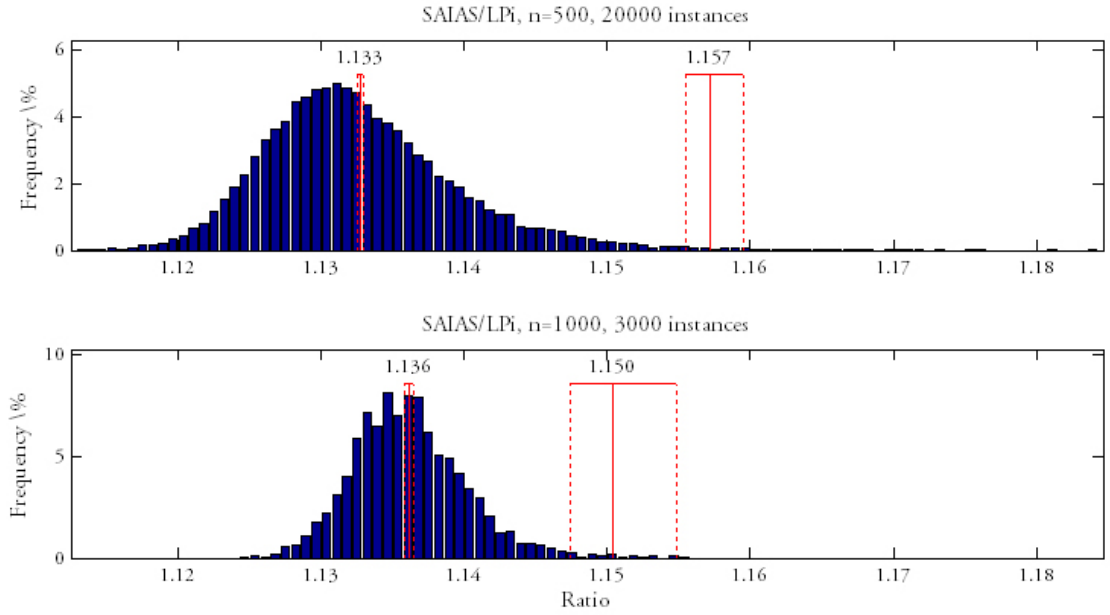


Figure 4.2: SAIAS/LPi Ratios with $n = 500$ and $n = 1,000$.

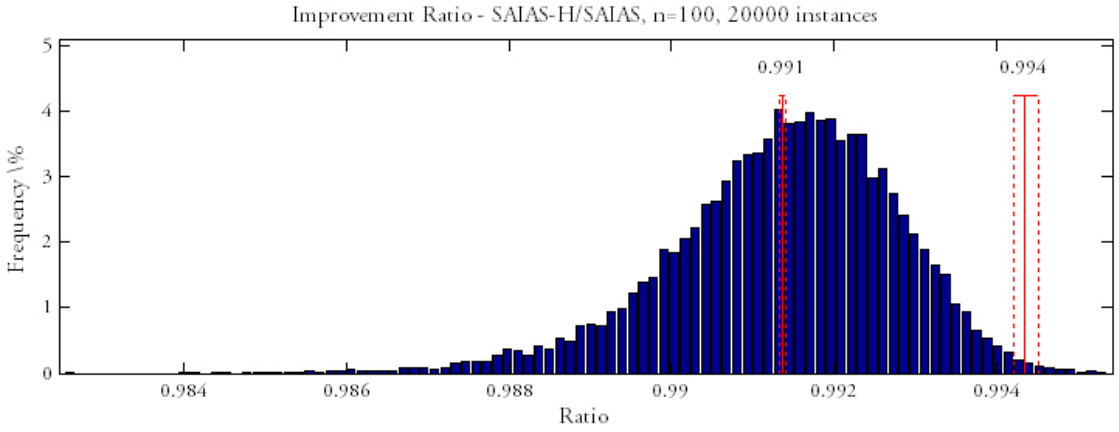


Figure 4.3: SAIAS-H Improvement Ratio, with $n = 100$.

to the relaxation as opposed to the algorithm's performance.

4.3.1.2 Heuristic Improvement and Online Setting.

Using the same parameters as in Section 4.3.1.1 we simulated 20,000 randomly generated instances and compared the empirical approximation ratios for the SAIAS algorithm and the

heuristic improvement, SAIAS-H. Figure 4.3 shows the improvement results.

As expected, the heuristic improvement reduces the approximation ratio on average by a factor of 0.991, and in 99.5% of the instances the improvement ratio was below 0.994 with 0.995 being the least improvement achieved. It is also important to note that the fraction of instances in which the heuristic algorithm computed the optimal solution increased from 0.05% to 0.13%.

Using this heuristic improvement we now study the performance of the SAIAS-H algorithm in the online setting. In the online setting we do not know any information of the jobs at time $t = 0$ as in the offline case, and we learn their size, weights, etc., only when they arrive. Hence, we can't compute the whole schedule beforehand. The way we adapt the SAIAS-H algorithm is as follows: we compute the approximate schedule using the SAIAS-H algorithm every time we finish a job, considering only the jobs present at that time, and every time a new job arrives, we recompute the speeds using the results of Section 4.3.1.1, but because preemption is not allowed, we only change the speeds and not the schedule. Figure 4.4 shows the empirical competitive ratios obtained over $20,000$ randomly generated instances, with all other settings as before.

In the online setting, the performance measure is called the competitive ratio, and is the ratio between the algorithm's output and the optimal offline solution. Figure 4.4 shows

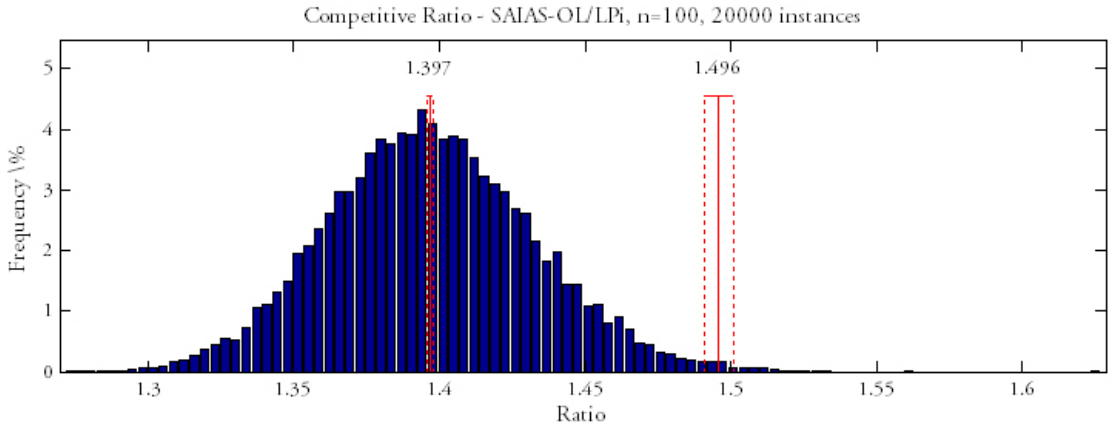
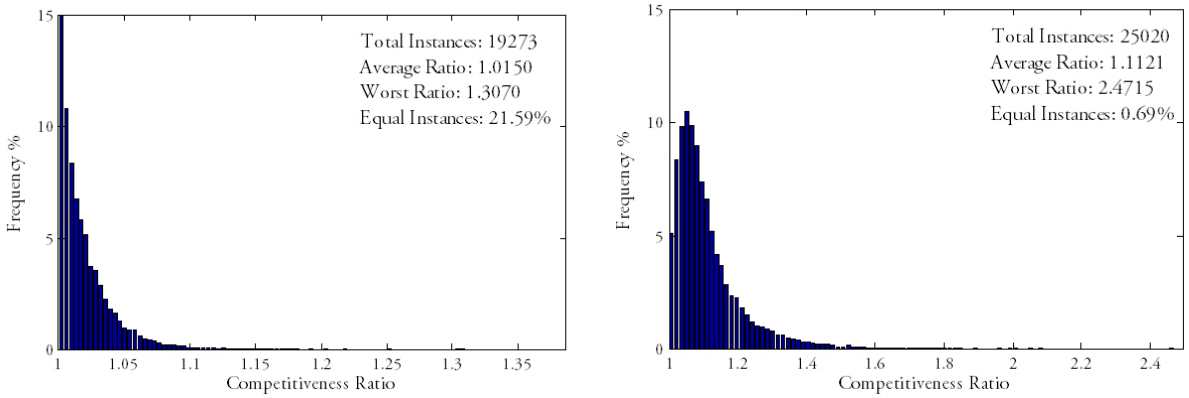


Figure 4.4: SAIAS-Online Competitive Ratio, with $n = 100$.



(a) $r_i \sim \text{unif}[0,\ 1.5\sum \frac{\rho_i}{\sigma_1}]$ (b) $r_i \sim \text{unif}[0,\ 0.1\sum \frac{\rho_i}{\sigma_1}]$

Figure 4.5: Competitive Ratios for the SAIAS-H Online Version

that the empirical competitive ratio for our algorithm. As expected, the competitive ratio is slightly larger than the approximation ratio of the offline case, but still it is very good, with an average of 1.397, 99.5% of the instances below 1.496 and a worst case of 1.534.

In the online setting, the selection of release dates constraints becomes crucial; if the upper bound in the interval defining the release date distribution is too large, the job queue will have just one job, and the online algorithm will be close to optimal. On the other hand, if the upper bound it is too close to 0, then the online algorithm reduces to an offline one, since all jobs are present at time $t = 0$. We found through simulations that, regardless of the number of jobs, the worst competitive ratios were obtained when the release dates were drawn from a $\text{unif}[0,\ 0.1\sum \frac{\rho_i}{\sigma_1}]$. Figure 4.5 shows histograms of the empirical competitive ratio for different release dates distributions.

4.3.1.3 Bimodal Job Sizes

One of the known bad instances for many scheduling problems is when we have several small jobs and then one large job arrives. In order to observe if such cases also present difficulty to our algorithms we simulated randomly generated instances where the jobs have unit size with probability p and or are of size $\rho >> 1$ with probability $1 - p$. Table 4.3 shows

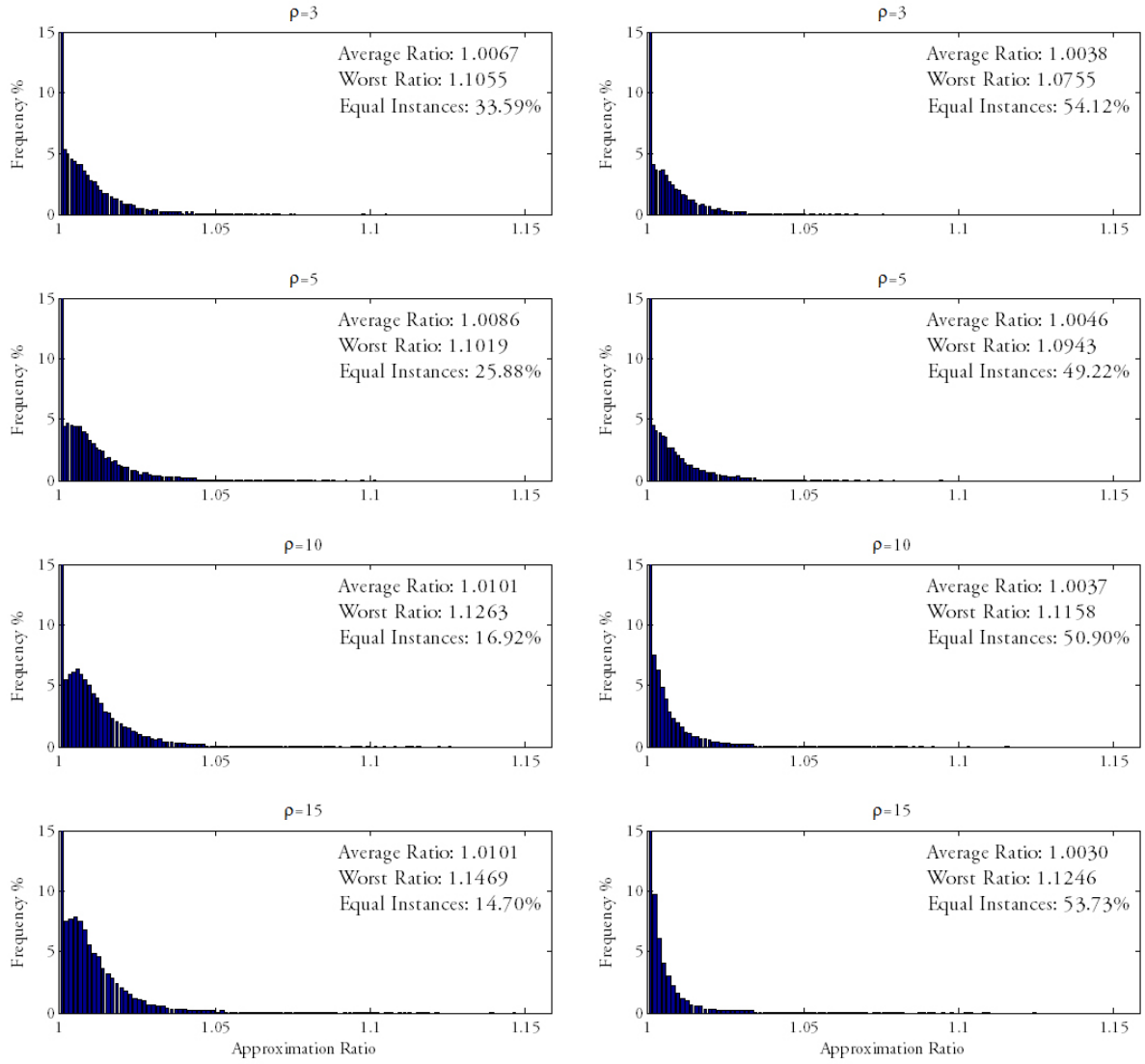
Problem	Algorithm	Average Ratio	Worst Ratio	Equal Instances
Bimodal Jobs with $\rho = 3$	SAIAS	1.0067	1.1055	33.59%
	SAIAS-H	1.0038	1.0755	54.12%
Bimodal Jobs with $\rho = 5$	SAIAS	1.0086	1.1019	25.88%
	SAIAS-H	1.0046	1.0943	49.22%
Bimodal Jobs with $\rho = 10$	SAIAS	1.0101	1.1263	16.92%
	SAIAS-H	1.0037	1.1158	50.90%
Bimodal Jobs with $\rho = 15$	SAIAS	1.0101	1.1469	14.70%
	SAIAS-H	1.0030	1.1246	53.73%

Table 4.3: Experimental Results in Bimodal Setting

a summary of the different cases analyzed and Figure 4.6 shows the histograms for both theSAIAS and SAIAS-H algorithms for different values of ρ for the case when $p = 0.7$ and $n = 6$. For each ρ, $20,000$ random instances were generated with the same settings as in the $1|prec|\sum E_i(s_i) + w_i C_i$ case before. These simulations show that although the approximation ratio varies with the size of the large job the values remain similar to what was obtained previously.

4.3.1.4 Sensitivity to Algorithm Parameters

Another important experimental result for both algorithms is their sensitivity towards the algorithm's parameters. Figure 4.7 shows the worst (dotted lines) and average (solid lines) approximation ratios for the SAIAS (in blue) and SAIAS-H (in red) algorithms, for various values of α. For each value of α, $30,000$ randomly generated instances with the same parameters as in the previous simulations, were analyzed. It is interesting to note that the worst approximation ratios achieve a minimum very close to where they do in theory (that is $\alpha \approx 0.5$ for the case with no release dates and $\alpha \approx \sqrt{2} - 1$ for the case with release dates), although the curves show that the algorithms are much less sensitive to this parameter than what shown in theory. For example in the case with no release dates the theoretical

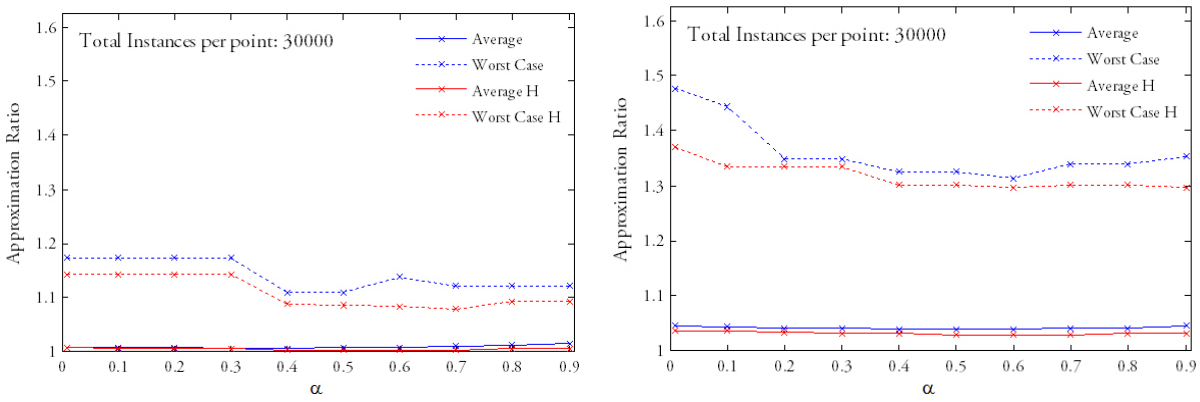


(a) SAIAS, $1|prec|\sum E_i(s_i) + w_iC_i$ setting (b) SAIAS-H, $1|prec|\sum E_i(s_i) + w_iC_i$ setting

Figure 4.6: SAIAS and SAIAS-H Comparison in Bimodal Setting

bound is approximately $4(1+\delta)(1+\epsilon) \approx 6.6$, whereas the empirical bound is close to 1.1. This is because the energy terms are able to absorb part of the error by modifying the speed accordingly, and when computing the theoretical bounds we rounded up several terms, something that is likely never going to happen in a real application.

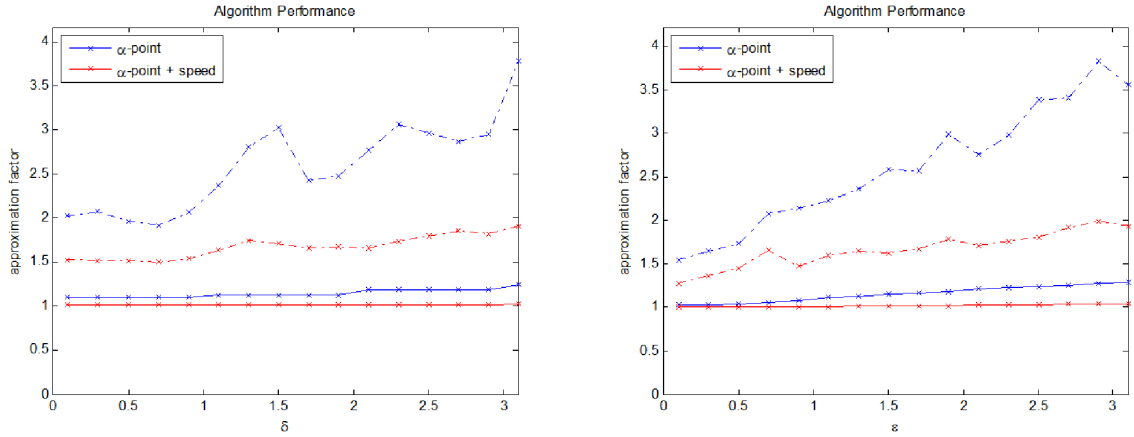
Similarly, we analyzed the effect of modifying δ and ϵ in both algorithms. Figure 4.8



(a) $1|prec|\sum E_i(s_i) + w_iC_i$ setting (b) $1|r_i, prec|\sum E_i(s_i) + w_iC_i$ setting

Figure 4.7: Sensitivity to α

shows the results for the sensitivity to changes in these two input variables. As expected, when the size of δ or ϵ increases, the empirical approximation ratio also increases. Also, the slope for the case when δ is increased (dotted blue line in Figure 4.8a) is smaller than when ϵ is increased (dotted blue line in Figure 4.8a), which is also expected since the rounding procedure increases the error closer to $(1 + \frac{\delta}{2}\delta)$. That is why both the worst case and average case approximation ratios increase slower when δ increases than when ϵ increases. Another



(a) $1|prec|\sum E_i(s_i) + w_iC_i$ setting (b) $1|prec|\sum E_i(s_i) + w_iC_i$ setting

Figure 4.8: Sensitivity to δ and ϵ

important thing to note is the improvement when using the speed heuristic. The sensitivity in this case is much smaller, specially in the case where δ is modified. This is because the error comes from the quantization of the speeds, whereas the heuristic fixes this by recomputing the optimal speeds. This also implies that the flexibility given by the speeds is really important, since the errors created by bad scheduling decisions can be absorbed by choosing the correct speeds.

4.3.1.5 Approximation Factor vs Instance Size

We made an additional set of simulations to check how the empirical approximation ratio varies with respect of the instance size. From the previous results apparently the ratio reduces when the size of the instance grows. In order to see if this happens we simulated for different values of n up to $20,000$ random generated instances.

Figure 4.9 shows the SAIAS/LPi ratios for the different values of n. The SAIAS/LPi ratios was used instead of the approximation ratio to make the simulations faster (and achievable in the case of larger values of n). In figure 4.9, the blue line shows the average ratio and the green one indicates the 99.5% proportion (i.e. 99.5% of the instances where below that value). Both of them have the 99.9% confidence interval around them. Finally, the red line shows the largest ratio among all the instances.

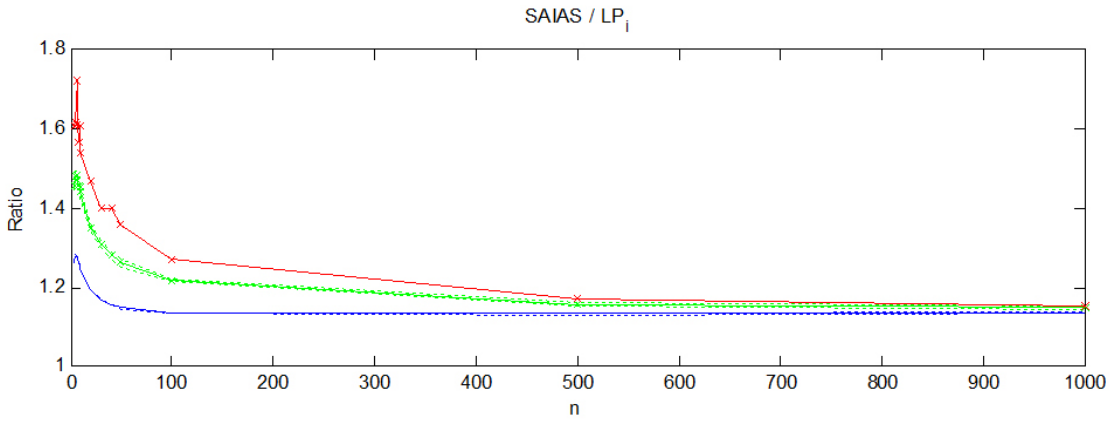


Figure 4.9: SAIAS/LPi Ratio for instances from $n = 4$ to $n = 1,000$

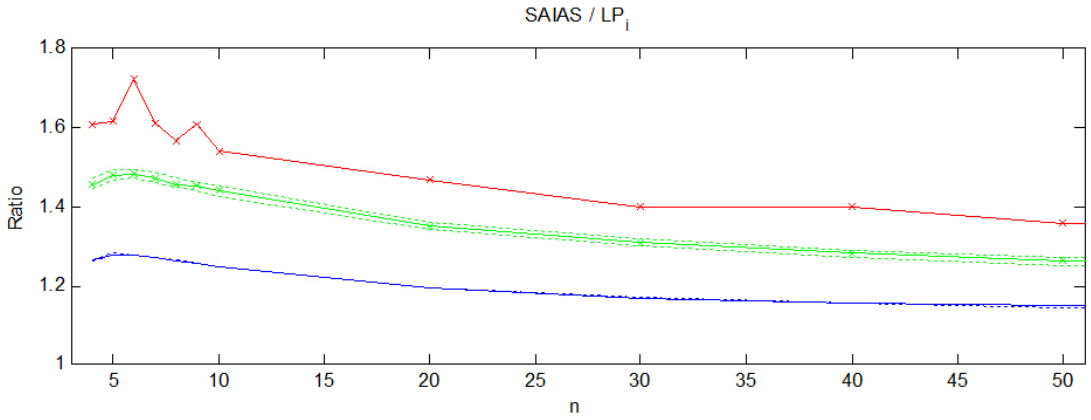


Figure 4.10: SAIAS/LPi Ratio for instances from $n = 4$ to $n = 50$

We can see that indeed the ratio reduces when the instance grows larger. The reason for this is, I believe, the flexibility of the speeds, which serve as a buffer to balance part of the error when the order is not optimal. Another observation is that there is a maximum in the empirical values around the value $n = 6$. Figure 4.10 shows a zoom for the initial part of figure 4.9, to show the maximum at $n = 6$.

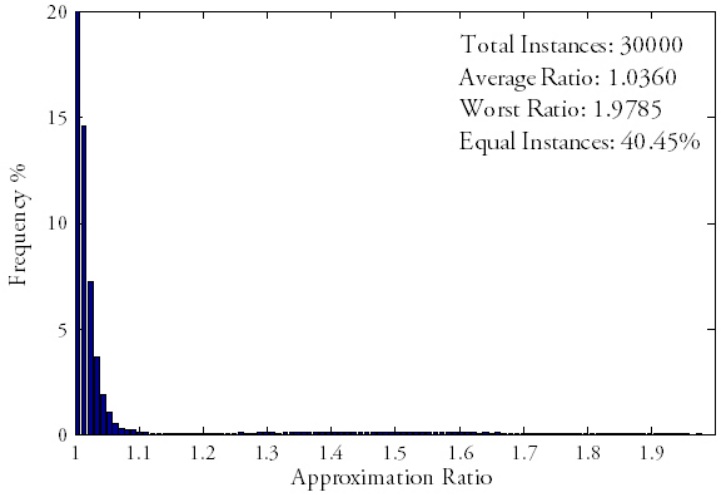


Figure 4.11: SAIAS with a different energy cost function.

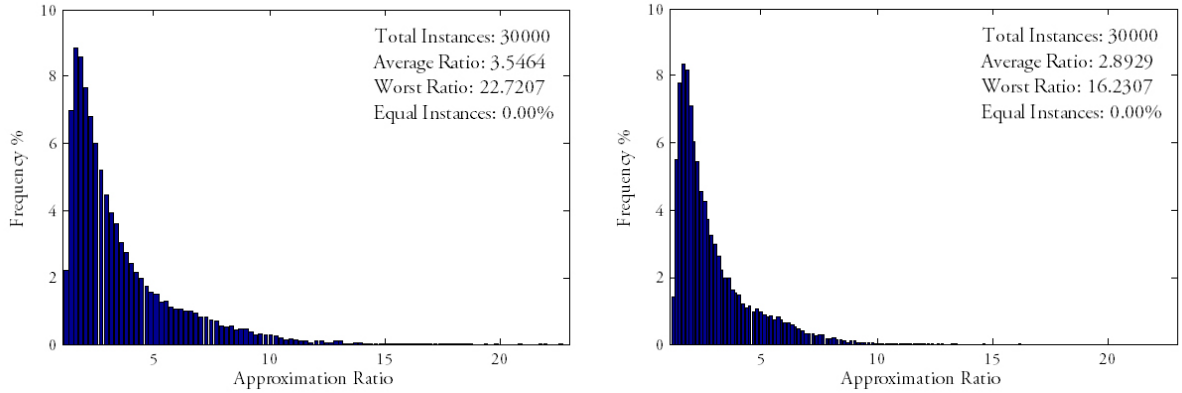
4.3.1.6 Other Energy Cost Functions

In order to show the performance of the SAIAS algorithm with other energy cost functions, we simulated the results using an energy cost function of the form $E_i(s_i) = a_i s^2 + b_i s + c_i$, where a_i, b_i, and c_i are chosen randomly. Figure 4.11 shows the approximation ratio histogram of $30,000$ randomly generated instances for the $1|prec|\sum E_i(s_i) + w_i C_i$ setting. Although the worst approximation ratio found is somewhat larger than in the standard case, the algorithm still performs very close to optimal, finding the optimal solution in over 40% of the instances.

4.3.2 Experimental Performance for Weighted Tardiness

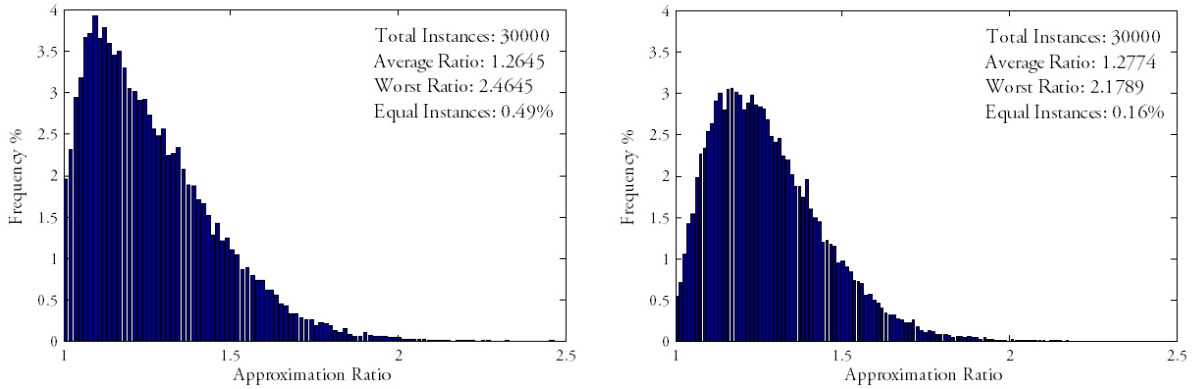
In this section we test the performance of the SAIAS-T algorithm similarly to what was done in Section 4.3.1. Figure 4.12a shows the approximation ratio histogram for the SAIAS-T algorithm, using $30,000$ randomly chosen instances with the same parameters as in Section 4.3.1, with $\alpha = \frac{1}{2}$. As expected the approximation ratios are worse, with an average ratio of 3.55 and worst ratio of 22.7, although the theoretical ratio is much higher: $4^{\beta}(1+\epsilon)^{\beta-1}(1+\delta)^{\beta-1} \approx 174$. As explained in Section 3.2 we cannot obtain theoretical worst case bounds for the SAIAS-T algorithm in the setting with release dates using the same methodology. The reason we cannot obtain the theoretical worst case bounds with our technique is because we rely on homogeneously speeding up jobs to ensure that they meet their deadlines, something that does not work when release dates are present. Still, we can simulate those instances and observe how bad the algorithm performs. Figure 4.12b shows the approximation ratio histogram for this setting.

It is interesting to note that, although the ratios are still very high, the performance of the algorithm improves vs. the case without release dates. We believe that the reason behind this effect is that when release dates are present the total weighted tardiness is higher and thus the total weighted tardiness is comparable in cost to the energy cost with higher speeds. This would validate our previous observation: we pay a significant price by speeding up jobs. To test this out we analyzed a modified version of the SAIAS-T algorithm, denoted SAIAS-T NS, in which we don't speed-up jobs, but keeping all the other settings as in the



(a) SAIAS-T for $1|prec|\sum E_i(s_i) + w_iT_i$ (b) SAIAS-T for $1|r_i, prec|\sum E_i(s_i) + w_iT_i$

Figure 4.12: Approximation Ratios for the SAIAS-T algorithm



(a) SAIAS-T, $1|prec|\sum E_i(s_i) + w_iT_i$ setting (b) SAIAS-T, $1|r_i, prec|\sum E_i(s_i) + w_iT_i$ setting

Figure 4.13: Approximation Ratios for the SAIAS-T NS algorithm

previous simulations.

Figure 4.13 shows the results for the setting with and without release dates. As expected the approximation ratios are much better, with average values below 1.3 and worst ratios below 2.5. Because we are not speeding-up jobs we are even able to find the optimal solution in a small fraction of the instances. This indicates that the speeding up procedure is really expensive in this case and that we might be able to find better ways of bounding the approximation ratios by taking the total costs as a whole and not separating the energy cost

and scheduling cost as we needed in our proofs.

In order to see the sensitivity of the algorithm to the speed-up factor γ, we simulated $20,000$ randomly generated instances and evaluated the output of the algorithm for several different values of $\gamma \in [0.1, \frac{(1+\epsilon)}{\alpha(1-\alpha)}]$. Figure 4.14 shows the results. We can see that the range of approximation ratios for each value of γ tested, with the average indicated by a red circle, and the 99.5% percentile indicated by a green asterisk The two green crosses indicate the 99.9% confidence interval for the 99.5% percentile. As observed previously, the best experimental speed-up ratio does not occur at $\gamma = \frac{(1+\epsilon)}{\alpha(1-\alpha)}$, but closer to $\gamma \approx 1$.

4.3.3 Experimental Performance for Weighted Flow Time

We also tested the performance of the algorithm with total weighted flow time scheduling metric. Most of the energy aware scheduling literature has focused in the non-weighted case, but because our algorithm allows jobs weights we analyzed the performance for this more general setting. In order to use the SAIAS algorithm in the $1|prec, r_i|\sum E_i(s_i) + w_i F_i$ setting, we only need to modify the cost function (3.49) to $\sum_{i,j,t} \left(E_i(\sigma_j) + w_i \left(\tau_{t-1} - r_i\right)\right) x_{ijt}$, and use the rest of the SAIAS algorithm, or its online version as before. Figure 4.15 shows the

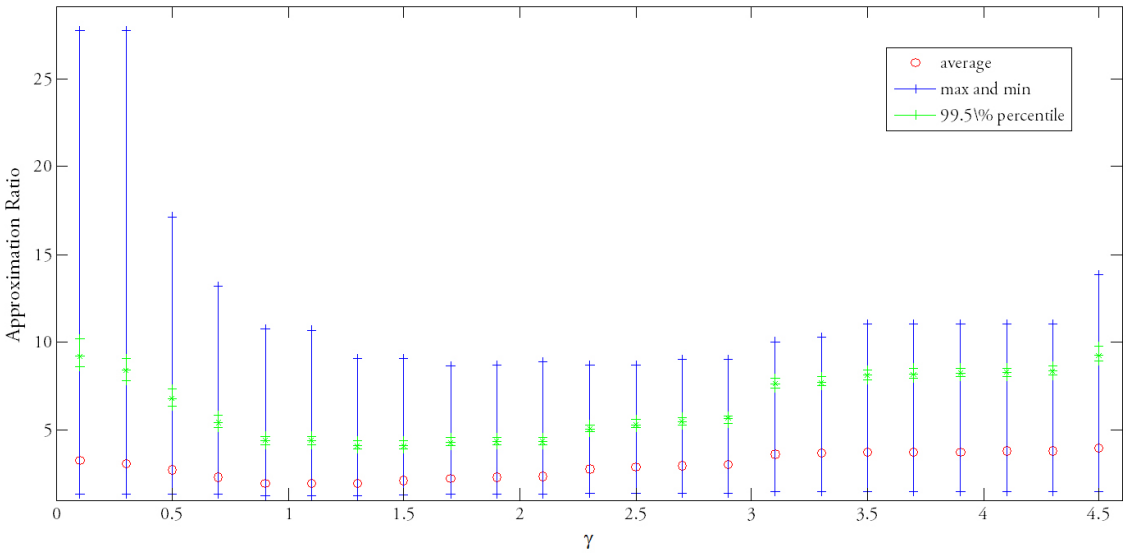


Figure 4.14: Approximation Ratios for Different Values of γ

results for 5,000 randomly generated instances, solving both the online and offline cases with $n = 100$ jobs. Since we are simulating large instances, we compare the algorithm's output with the LPi bound, as opposed to the IPt solution. All the other simulation settings were the same as in the previous sections.

The first histogram on Figure 4.15 shows the empirical approximation ratio in the offline setting. The average approximation ratio was 1.500, with 99.5% quantile given by 2.012, and a worst case of 2.334. Note that the real approximation ratios should be smaller when compared to the IPt solution. The second histogram shows the empirical competitive ratio in the online setting. In this case the values are much higher since the algorithm only knows the size and associated weight of the jobs present at each point in time, resulting in an average competitive ratio of 2.590, with 99.5% quantile given by 3.204, and a worst case of 3.413. The difference with the offline setting can be considered as the benefit from knowing the size, weight, and release date information of all jobs beforehand.

The summary of the results for this setting, displayed in Table 4.4, shows that the algorithm performs very well in practice both in the online and offline cases.

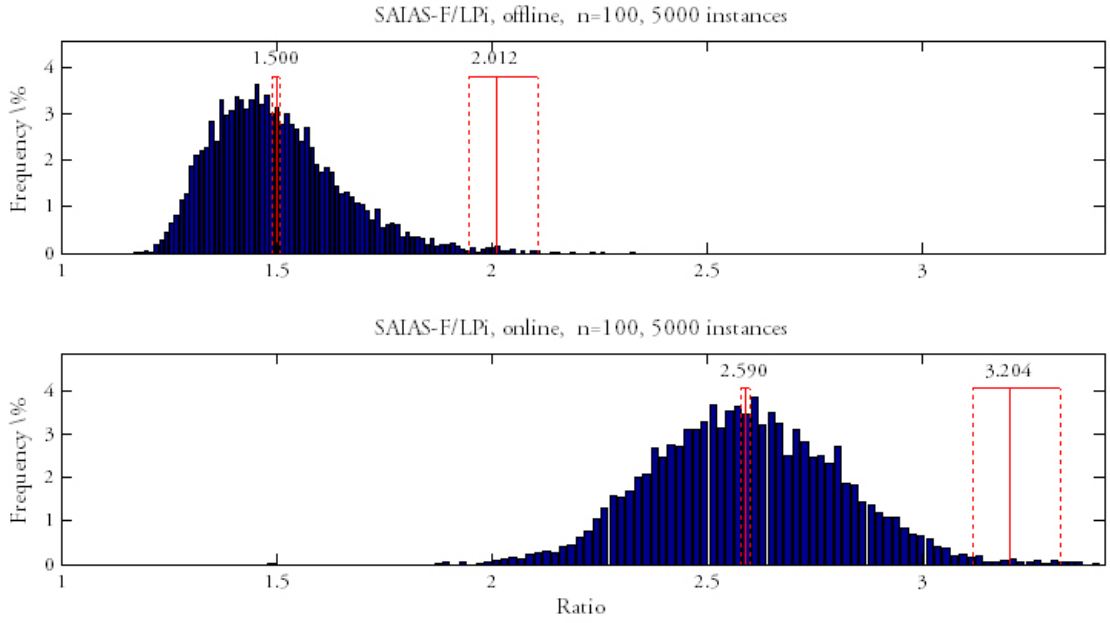


Figure 4.15: SAIAS-F Offline and Online Ratios, with $n = 100$.

Problem	Instances	Size (n)	Average Ratio	99.5%	Worst Ratio
Offline	$5,000$	100	1.500*	2.012	2.334
Online (no *prec*)	$5,000$	100	2.590*	3.204	3.413

Table 4.4: Experimental Results Summary for Total Weighted Flow Time

4.3.4 Experimental Performance for Multiple Machines

In this section we present the experimental results of two extensions of the SAIAS algorithm: the online setting and the multiple parallel machines setting.

In this section we consider the $P|r_i|\sum E_i(s_i) + w_iC_i$ problem, with q machines running in parallel,

we not only have to choose the sequence and speed, but also the machine on which to run the job. We extended the SAIAS-H algorithm to parallel machines, as follows. We first extend the LP (3.49)-(3.54) for multiple parallel machines, i.e. we have variables of the form x_{ijkt} that represent the fraction of job i that runs at speed σ_j on machine k and it is finished by time interval I_t, and we modify the constraints accordingly. Then $\tilde{x}_{ik} = \sum_{jt} x_{ijkt}$ represents the fraction of job i processed at machine k. SAIAS-P in Figure 4.4 describes the algorithm for parallel machines.

The simulation setting is similar to the previous sections, but in this case we increased the number of jobs to $n = 15$ and set $q = 4$. Since computing the exact optimal solution of the corresponding very large IP is computationally expensive, only $3,700$ instances were simulated. Figure 4.16a shows a histogram of the approximation ratios found through the

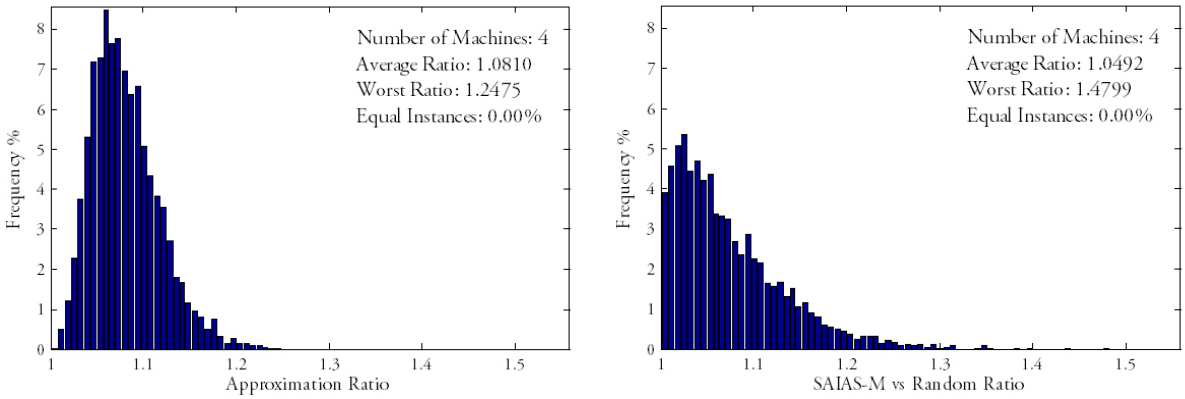
Algorithm 4.4 SAIAS-P

1 Solve the extended LP and compute $\tilde{x}_{ik}$.

2 Assign job i to machine $k_i = \arg\max_k\{\tilde{x}_{ik}\}$

3 For each machine k apply the SAIAS-H algorithm for the selected jobs.

4 **return** the speeds and schedules for each machine.



(a) SAIAS-P (b) Ratio vs Randomly Choosing Machines

Figure 4.16: Approximation Ratios for Parallel Machines

simulations. The average ratio is still very close to optimal, and even the worst ratio found is quite small. However, in none of the instances was the algorithm's output equal to the exact optimal solution.

Since speed scaling, as we have shown in previous sections, is known to significantly improve performance, we also computed the approximation ratio for random assignment to machines, in order to evaluate the added value of the algorithm. Figure 4.16 shows the ratio between the solution given by the SAIAS-P algorithm and the solution found by randomly choosing a machine for that same instance. The fact that this ratio is always larger than 1 indicates that the SAIAS-P algorithm actually helps in selecting the machine.

Chapter 5

Resource Cost Aware Scheduling

MANAGING non-renewable resource consumption is fast emerging as a problem of critical importance. There is always a trade-off between resource consumption and performance: more resource consumption typically results in better performance. This trade-off also arises in many scheduling problems, where resource management decisions must be combined with the scheduling decisions to optimize a global objective.

Recently, scheduling problems in which one has to trade scheduling performance using metrics such as completion time, tardiness, or flow time, with CPU speed, and therefore the energy consumed, have been extensively studied. However, as discussed in Chapter 2, the problem of balancing resource consumption with scheduling performance was proposed much earlier. Vickson [Vickson, 1980] observed that in many practical settings, the processing time of a job depends on the amount of resources (e.g. catalizer, workforce size, energy, etc.) utilized, and the relationship between resource utilization and processing time depends on each job's characteristics. Other examples of scheduling problems with resource dependent job processing time include repair and maintenance processes [Duffuaa *et al.*, 1999]; ingot preheating processes in steel mills, where the batches need to be scheduled and the amount of gas used and the concentration level determine the time required to preheat the ingots [Janiak, 1991; Williams, 1985]; many workforce intensive operations; VLSI circuit design [Monma *et al.*, 1990]; and more recently processing tasks in a CPU, where the job processing times depends

on CPU speed, the available RAM, bus speed, as well as other system resources.

Given that our previous results allow for very general functions, in this chapter we further extend the problem formulated in Chapter 3 to the more general setting of resource cost aware scheduling when using total weighted completion time as the scheduling metric. Thus, we bridge the gap between the Resource Dependent Job Processing Time literature and the newer Energy Aware Scheduling literature, generalizing both of them.

We consider the non-preemptive, offline problem on one machine, and allow arbitrary precedence constraints and arbitrary release dates as well. Our objective is to minimize the sum of our scheduling metric and the total resource consumption cost.

We make several contributions to the problem of scheduling with non-renewable resources

- We introduce a model that extends the previous cost models (linear, convex, and other energy models) by allowing a more general relation between job processing time (or equivalent processing speed) and resource consumption.
- We further generalize the problem by allowing arbitrary precedence constraints and release dates.
- We give approximation algorithms for minimizing an objective that is a combination of a scheduling metric (weighted completion time) and resource consumption cost.

We consider a more general model of resource cost than has previously been used. As noted in Chapter 2 the resource dependent job processing time literature either focuses on job's processing times that depend linearly on resource consumption or a convex relation of the form $(\rho_i/u_i)^k$, generally considering only a single resource. Our setting captures both of these models by considering an arbitrary non-negative speed function $\mathcal{S}(\mathbf{\Psi}^{(i)})$, where $\mathbf{\Psi}^{(i)} \in \mathbf{\Psi} = \{\mathbf{\Psi}^{(1)}, \ldots, \mathbf{\Psi}^{(q)}\}$ denotes one of the q allowable operating points of the resources. We also generalize the resource cost, which is generally linear in the literature, by considering an arbitrary non-negative job-dependent resource cost function $\mathcal{R}_i(\mathbf{\Psi}^{(i)})$.

This chapter contains results for two related scheduling problems, we state here the most general result:

Theorem 5.1. *Given n jobs with precedence constraints and release dates and a general non-negative resource cost function, there is an $O(1)$-approximation algorithm for the problem of non-preemptively minimizing a weighted sum of the completion time and resource cost.*

The constants in the $O(1)$ are modest. Given some $\epsilon > 0$, the algorithm has a $(4+\epsilon)$-approximation ratio when only precedence constraints exist, and $(3+2\sqrt{2}+\epsilon)$-approximation ratio when release dates are added.

In this chapter, we extend the interval-indexed IP described in Chapter 3 to handle resource costs and speed scaling, and then modify the α-point based rounding algorithm presented in Chapter 3 to obtain the resulting schedules. We assume in Sections 5.1 and 5.2 that we have a discrete set of q allowable resource operating points $\mathbf{\Psi} = \{\mathbf{\Psi}^{(1)}, \ldots, \mathbf{\Psi}^{(q)}\}$, and that the speed at which the job is precessed is a general non-negative function of the resource operating point. We will describe only the algorithm using the interval-indexed linear programs in this chapter since the time-indexed one is similar to what was described in Chapter 3. In our interval-indexed IP, a variable x_{ijt} is 1 if job i runs at resource operating point $\mathbf{\Psi}^{(j)}$ and completes in interval t. We can then extend the standard interval-indexed integer programming formulation to take the extra dimensions of resource consumption and speed into account (see Section 5.1 for details). Once we have solved its linear program relaxation (LPi), we need to determine *both an α-point and α-speed*. The key insight is that by "summarizing" each dimension appropriately, we are able to make the correct choice for the other dimension. At a high level, we first choose the α-point by "collapsing" all pieces of a job that completes in the LPi in interval t (these pieces have different speeds), being especially careful with the last interval, where we may have to choose only some of the speeds. We then use *only* the pieces of the job that complete before the α-point to choose the speed, where the speed is chosen by collapsing the time dimension and then interpreting the result as a probability mass function (pmf), where the probability that the job is run at speed $\mathcal{S}(\mathbf{\Psi}^{(j)})$ depends on the total amount of processing done at that operating point. We then use the concept of α-speeds, which is related to the expected value under this pmf, and run the job at this speed (see Section 5.2 for more details). We combine this rounding method

with extensions of the more traditional methods for dealing with precedence constraints and release dates to obtain our algorithm.

5.1 Problem formulation.

We are given a single machine that requires p different resources, indexed $1, \ldots, p$ to run. As mentioned in the previous chapter, examples of these resources include energy, fuel, maintenance level, wear rate, reaction catalizer, and workforce size among others. The machine has q different resource operating points $\mathbf{\Psi}^{(i)} \in \mathbf{\Psi} = \{\mathbf{\Psi}^{(1)}, \ldots, \mathbf{\Psi}^{(q)}\}$, where $\mathbf{\Psi}^{(i)} = \begin{bmatrix} \Psi_1^{(i)} & \ldots & \Psi_p^{(i)} \end{bmatrix}$ is described by a vector of p values, one for each resource. We are also given a function $\mathcal{S} : \mathbb{R}^p \to \mathbb{R}_+$ which maps each operating point $\mathbf{\Psi}^{(j)}$ to a speed $\sigma_j = \mathcal{S}(\mathbf{\Psi}^{(j)})$, and a function $\mathcal{R}_i(\boldsymbol{\psi}^{(i)})$, with $\mathcal{R} : \mathbb{R}^p \to \mathbb{R}_+$, which denotes the cost of running job i at the resource operating point $\boldsymbol{\psi}^{(i)}$. Additionally, we are given n jobs, where job i has a processing requirement of ρ_i machine cycles, a release time r_i, and an associated positive weight w_i. We may also be given precedence constraints among the jobs but we do not allow preemption.

A *schedule* defines, for each job, a time interval during which it runs, and for each time in that interval, a resource operating point from the allowable set. As in previous work, we can make some observations that simplify the structure of a schedule. By time sharing between different operating points the machine can run at any point within the convex hull of $\mathbf{\Psi}$. We thus extend the domain of the speed function and the cost function to include points ψ in the convex hull of $\mathbf{\Psi}$ in the natural way: for $\boldsymbol{\psi}^{(i)}$ such that $\boldsymbol{\psi}^{(i)} = \sum_{j=1}^{q} \lambda_j \mathbf{\Psi}^{(j)}$, with $\sum_j \lambda_j = 1$ and $\lambda_j \in [0, 1]$, then if $\psi = \sum_j \delta_j \Psi^{(j)}$ then $\mathcal{S}(\psi) = \sum_j \delta_j \mathcal{S}(\Psi^{(j)})$ and $\mathcal{R}_i(\boldsymbol{\psi}^{(i)}) = \sum_{j=1}^{q} \lambda_j \mathcal{R}_i(\mathbf{\Psi}^{(j)})$. Thus, by extending our domain in this way, we can assume that each job runs at one resource operating point, and one speed. We can further assume that a point with lower speed also has lower cost, for otherwise we could achieve that point by running at a higher speed and then idling, thereby achieving an even better cost. Throughout the paper, we will use capital Ψ to denote the input set of operating points and lowercase ψ

to denote points in the convex hull.

With the above extension, we can define a schedule precisely as follows. Let $\boldsymbol{\psi}^{(i)}$ denote the operating point at which job i runs, thus $s_i = \mathcal{S}(\boldsymbol{\psi}^{(i)})$ denotes the speed at which job i runs in the machine, and $p_i = \frac{\rho_i}{\mathcal{S}(\boldsymbol{\psi}^{(i)})}$, its processing time. Let C_i denote the completion time of job i, and let $\Pi = \{\pi(1), \ldots, \pi(n)\}$ denote the order in which the jobs are processed, i.e. $\pi(k) = i$ implies that job i is the k-th job to be processed. Then $C_{\pi(i)} = \max\{r_{\pi(i)}, C_{\pi(i-1)}\} + \frac{\rho_{\pi(i)}}{s_{\pi(i)}}$ is the completion time of the i-th job to be processed, with $C_{\pi(0)} = 0$.

The objective is to compute a feasible schedule, consisting of an order Π, possibly subject to precedence and/or release date constraints, and the vector of resource requirements $\boldsymbol{\psi} = \left[\boldsymbol{\psi}^{(1)} \quad \ldots \quad \boldsymbol{\psi}^{(n)}\right]$ that minimizes the total cost,

$$f(\Pi, \boldsymbol{\psi}) = \sum_{i=1}^{n} \left[\mathcal{R}_i(\boldsymbol{\psi}^{(i)}) + w_{\pi(i)} C_{\pi(i)}\right]. \tag{5.1}$$

We assume, w.l.o.g., that the resource operating points are ordered by speed (slowest first), and use $\sigma_i = \mathcal{S}(\boldsymbol{\Psi}^{(i)})$ to denote the i^{th} slowest speed. Note that since any speed used is a convex combination of these σ_i's, we will never consider any speed slower than σ_1 or faster than σ_q (which we denote by $\sigma_{\max}$).

5.1.1 Interval-indexed formulation.

We now modify and extend the interval-indexed formulation proposed in Chapter 3 to the general resource cost functions.

The interval-indexed formulation divides the time horizon into geometrically increasing intervals, and the completion time of each job is assigned to one of these intervals. Since the completion times are not associated to a specific time, the completion times are not precisely known but are lower bounded. By controlling the growth of each interval one can obtain a sufficiently tight bound.

The problem formulation is as follows. We divide the time horizon into the following geometrically increasing intervals: $[\kappa, \kappa]$, $(\kappa, (1+\epsilon)\kappa]$, $((1+\epsilon)\kappa, (1+\epsilon)^2\kappa]$, ..., where $\epsilon > 0$ is an arbitrary small constant, and $\kappa = \frac{\rho_{\min}}{\sigma_{\max}}$ denotes the smallest interval size that will hold at

least one whole job. We define interval $I_t = (\tau_{t-1}, \tau_t]$, with $\tau_0 = \kappa$ and $\tau_t = \kappa(1+\epsilon)^{t-1}$. The interval index ranges over $\{1, \ldots, T\}$, with $T = \min\{\lceil t \rceil : \kappa(1+\epsilon)^{t-1} \geq \max_{i=1}^{n} r_i + \sum_{i=1}^{n} \frac{\rho_i}{\sigma_1}\}$; and thus, we have a polynomial number of indices t.

Let

$$x_{ijt} = \begin{cases} 1, & \text{if job } i \text{ runs at o.p. } \mathbf{\Psi}^{(j)} \text{ and completes in time interval } I_t \\ 0, & \text{otherwise.} \end{cases} \tag{5.2}$$

By using the lower bounds τ_{t-1} of each time interval I_t, a lower bound to (5.1) is written as,

$$\min_{\mathbf{x}} \sum_{i=1}^{n} \sum_{j=1}^{q} \sum_{t=1}^{T} \left(\mathcal{R}_i(\mathbf{\Psi}^{(j)}) + w_i \tau_{t-1} \right) x_{ijt}. \tag{5.3}$$

The following are the constraints required for the $1|r_i, prec| \sum \mathcal{R}_i(\boldsymbol{\psi}^{(i)}) + w_i C_i$ problem:

1. Each job must finish in a unique time interval and speed; therefore for $i = \{1, \ldots, n\}$:

$$\sum_{j=1}^{q} \sum_{t=1}^{T} x_{ijt} = 1. \tag{5.4}$$

2. Since only one job can be processed at any given time, the total processing time of jobs up to time interval I_t must be at most τ_t units. Thus, for $t = \{1, \ldots, T\}$:

$$\sum_{i=1}^{n} \sum_{j=1}^{q} \sum_{u=1}^{t} \frac{\rho_i}{\sigma_j} x_{iju} \leq \tau_t. \tag{5.5}$$

3. Job i running at speed σ_j requires $\frac{\rho_i}{\sigma_j}$ time units to be processed, and considering that its release time is r_i, then for $i = \{1, \ldots, n\}$, $j = \{1, \ldots, q\}$, and $t = \{1, \ldots, T\}$:

$$x_{ijt} = 0, \quad \text{if } \tau_t < r_i + \frac{\rho_i}{\sigma_j}. \tag{5.6}$$

4. For $i = \{1, \ldots, n\}$, $j = \{1, \ldots, q\}$ and $t = \{1, \ldots, T\}$:

$$x_{ijt} \in \{0, 1\}. \tag{5.7}$$

5. The precedence constraint $i_1 \prec i_2$ implies that job i_2 cannot finish in an interval earlier than i_1. Therefore for every $i_1 \prec i_2$ constraint we have that for $t = \{1, \ldots, T\}$:

$$\sum_{j=1}^{q}\sum_{u=1}^{t} x_{i_1ju} \geq \sum_{j=1}^{q}\sum_{u=1}^{t} x_{i_2ju}. \tag{5.8}$$

Just like in Chapter 3, it is important to note that this integer program only provides a lower bound for (5.1); in fact its optimal solution may not be schedulable, since constraints (5.5) do not imply that only one job can be processed at a single time, they only bound the total amount of work in $\cup_t I_t$.

5.2 Approximation algorithm for weighted completion time.

We now describe the approximation algorithm for the weighted completion time, called Schedule by α-intervals and α-speeds (SAIAS), for the setting with resource costs. Algorithm 5.1 details the procedure.

Let $\bar{x}_{ijt}$ denote the optimal solution of the linear relaxation of the integer program (5.3)-(5.8), in which we change constraints (5.7) for $x_{ijt} \geq 0$. In step 3 of the algorithm we compute the optimal solution $\bar{\mathbf{x}}$ and in step 4, given $0 \leq \alpha \leq 1$, we compute the α-interval I_i^α of job i, defined as

$$I_i^\alpha = \min\left\{\tau : \sum_{j=1}^{q}\sum_{u=1}^{\tau} \bar{x}_{iju} \geq \alpha\right\}. \tag{5.9}$$

Since several jobs may finish in the same interval, let J_t denote the set of jobs that finish in interval I_t, $J_t = \{i : I_i^\alpha = t\}$, and we use these sets to determine the order Π^α as described in step 5.

Next, in step 6, we compute the α-speeds as follows. Since $\sum_{j=1}^{q}\sum_{u=1}^{I_i^\alpha} \bar{x}_{iju} \geq \alpha$, we define

Algorithm 5.1 SCHEDULE BY α-INTERVALS AND α-SPEEDS FOR RESOURCE COSTS (SAIAS-RC)

Inputs: set of jobs, $\alpha \in (0,1)$, $\epsilon > 0$, set of resource operating points $\mathbf{\Psi}$, speed function S and resource function R.

1 Divide time into increasing time intervals $I_t = (\tau_{t-1}, \tau_t]$, with $\tau_t = \kappa(1+\epsilon)^{t-1}$.

2 Compute the set of possible speeds $\mathbf{S} = \{\sigma_1, \ldots, \sigma_q\}$.

3 Compute an optimal solution $\bar{\mathbf{x}}$ to the linear relaxation (5.3)-(5.8).

4 Compute the α-intervals $\mathbf{I}^\alpha$ and the sets J_t. via (5.9)

5 Compute an order Π^α that has sets J_t ordered in non-decreasing values of t and the jobs within each set in a manner consistent with the precedence constraints.

6 Compute the α-speeds $\mathbf{s}^\alpha$ via (5.12).

7 Set the i-th job to start at time $\max\{r_{\pi(i)}, C^\alpha_{\pi(i-1)}\}$, where $C^\alpha_{\pi(i-1)}$ is the completion time of the previous job using the rounded α-speeds, and $C^\alpha_{\pi(0)} = 0$.

8 **return** speeds $\mathbf{s}^\alpha$, order Π^α, and completion times $\bar{\mathbf{C}}^\alpha$.

auxiliary variable $\{\tilde{x}_{ijt}\}$ as:

$$\tilde{x}_{ijt} = \begin{cases} \bar{x}_{ijt}, & t < I_i^\alpha \\ \max\left\{\min\left\{\bar{x}_{ijI_i^\alpha}, \alpha - \sum_{l=1}^{j-1} \bar{x}_{ilI_i^\alpha} - \beta_i\right\}, 0\right\}, & t = I_i^\alpha \\ 0, & t > I_i^\alpha, \end{cases} \tag{5.10}$$

where $\beta_i = \sum_{j=1}^{q} \sum_{u=1}^{I_i^\alpha - 1} \bar{x}_{iju} < \alpha$. Note that for this auxiliary variable, we have that $\sum_{j=1}^{q} \sum_{u=1}^{I_i^\alpha} \tilde{x}_{iju} = \alpha$. This is a key step that allows us to truncate the fractional solution so that for every job i, the sum of $\tilde{x}_{ijt}$ up to time interval I_i^α for each speed j can be interpreted as a probability mass function. We define this probability mass function (pmf) $\boldsymbol{\mu}^i = (\mu_1^i, \ldots, \mu_q^i)$ on the set of speeds $\mathbf{S} = \{\sigma_1, \ldots, \sigma_q\}$ as

$$\mu_j^i = \frac{1}{\alpha} \sum_{u=1}^{I_i^\alpha} \tilde{x}_{iju}. \tag{5.11}$$

Let $\hat{s}_i$ define a random variable distributed according to the pmf $\boldsymbol{\mu}^i$, i.e. $\mu_j^i = \mathbb{P}(\hat{s}_i = \sigma_j)$.

Then, the α-speed of job i, s_i^α, is defined as follows:

$$\frac{1}{s_i^\alpha} = \mathbb{E}\left[\frac{1}{\hat{s}_i}\right] = \sum_{j=1}^{q} \frac{\mu_j^i}{\sigma_j} \Rightarrow s_i^\alpha = \frac{1}{\mathbb{E}\left[\frac{1}{\hat{s}_i}\right]}. \tag{5.12}$$

We define the α-speeds using the reciprocal of the speeds since the completion times are proportional to the reciprocals, and we need to bound completion times in the analysis of the algorithm. Note that (5.11) defines the fraction of the machine cycles requirement ρ_i that must be processed at each operating point $\mathbf{\Psi}^{(j)}$ to achieve the α-speed s_i^α.

Finally, in steps 7 and 8 we compute the completion times given the calculated speeds and return the set of speeds $\mathbf{s}^\alpha$, the order Π^α and the completion times $\mathbf{C}^\alpha$.

We now analyze this algorithm's performance both with and without release dates. In the following subsections we will assume w.l.o.g. that $I_1^\alpha \leq I_2^\alpha \leq \ldots I_n^\alpha$.

5.2.1 Single machine problem with precedence constraints.

In this section, we analyze our algorithm for the case of precedence constraints but no release dates. We first prove that the output of the SAIAS algorithm is indeed feasible.

Lemma 5.1. *Suppose $i_1 \prec i_2$. Then (5.8) implies that $I_{i_1}^\alpha \leq I_{i_2}^\alpha$.*

Proof. Evaluating the LP constraint (5.8) corresponding to $i_1 \prec i_2$, for $t = I_{i_2}^\alpha$, we have that,

$$\sum_{j=1}^{q} \sum_{u=1}^{I_{i_2}^\alpha} x_{i_1 j u} \geq \sum_{j=1}^{q} \sum_{u=1}^{I_{i_2}^\alpha} x_{i_2 j u} \geq \alpha, \tag{5.13}$$

where the last inequality follows from the definition of $I_{i_2}^\alpha$. The chain of inequalities implies that

$$\sum_{j=1}^{q} \sum_{u=1}^{I_{i_2}^\alpha} x_{i_1 j u} \geq \alpha,$$

so $I_{i_1}^\alpha \leq I_{i_2}^\alpha$. □

Since the SAIAS algorithm schedules jobs by first ordering the sets J_t in increasing order of t, and then orders the jobs within each set in a way that is consistent with the

precedence constraints, Lemma 5.1 implies that the SAIAS algorithm preserves the precedence constraints, and, therefore, the output of the algorithm is feasible.

Theorem 5.2. *The SAIAS algorithm with $\alpha = \frac{1}{2}$ is a $(4+\epsilon)$-approximation algorithm for the $1|prec|\sum \mathcal{R}_i(\boldsymbol{\psi}^{(i)}) + w_i C_i$ problem, with a general non-negative $\mathcal{R}_i(\boldsymbol{\psi})$ resource cost function.*

Proof. Let x^*_{ijt} denote an optimal solution to the integer problem (5.3)-(5.8), $\bar{x}_{ijt}$ the fractional solution of its linear relaxation, and $\tilde{x}_{iju}$ the auxiliary variables calculated for the SAIAS algorithm.

Since in (5.3) the completion time for jobs completed in interval I_t is τ_{t-1}, it follows that,

$$\sum_{i=1}^{n}\sum_{j=1}^{q}\sum_{t=1}^{T}\left(\mathcal{R}_i(\boldsymbol{\Psi}^{(j)}) + w_i\tau_{t-1}\right)\bar{x}_{ijt} \leq f(\Pi^*, \boldsymbol{\psi}^*). \tag{5.14}$$

Let $\boldsymbol{\psi}^{(i)\alpha} = \sum_{j=1}^{q}\mu_j^i\boldsymbol{\Psi}^{(j)}$ denote the effective operating point that achieves the required α-speed, and $\hat{\boldsymbol{\psi}}^{(i)}$ define a random variable distributed according to the pmf $\boldsymbol{\mu}^i$, just like with $\hat{s}_i$. The resource cost terms of the algorithm's solution are bounded as follows,

$$\mathcal{R}_i(\boldsymbol{\psi}^{(i)\alpha}) = \mathcal{R}_i(\mathbb{E}\left[\hat{\boldsymbol{\psi}}^{(i)}\right]) \leq \mathbb{E}\left[\mathcal{R}_i(\hat{\boldsymbol{\psi}}^{(i)})\right] = \sum_{j=1}^{q}\mu_j^i\mathcal{R}_i(\boldsymbol{\Psi}^{(j)}), \tag{5.15}$$

where the inequality follows from Jensen's Inequality applied to the convex function $\mathcal{R}_i()$. Using the definition of μ_j^i in (5.11) and given that $0 \leq \alpha \leq 1$, $\epsilon > 0$, and $\tilde{x}_{ijt} \leq \bar{x}_{ijt}$, it follows that,

$$\mathcal{R}_i(\boldsymbol{\psi}^{(i)\alpha}) \leq \frac{1}{\alpha}\sum_{j=1}^{q}\sum_{u=1}^{I_i^\alpha}\mathcal{R}_i(\boldsymbol{\Psi}^{(j)})\tilde{x}_{iju} \leq \frac{(1+\epsilon)}{\alpha(1-\alpha)}\sum_{j=1}^{q}\sum_{u=1}^{T}\mathcal{R}_i(\boldsymbol{\Psi}^{(j)})\bar{x}_{iju}. \tag{5.16}$$

Since there are no release date constraints there is no idle time between jobs and thus,

$$C_i^\alpha = \sum_{j=1}^{i}\frac{\rho_j}{s_j^\alpha} = \sum_{j=1}^{i}\rho_j\mathbb{E}\left[\frac{1}{\hat{s}_j}\right] = \frac{1}{\alpha}\sum_{j=1}^{i}\sum_{l=1}^{q}\sum_{u=1}^{I_j^\alpha}\frac{\rho_j}{\sigma_l}\tilde{x}_{jlu} \leq \frac{1}{\alpha}\sum_{j=1}^{n}\sum_{l=1}^{q}\sum_{u=1}^{I_i^\alpha}\frac{\rho_j}{\sigma_l}\bar{x}_{jlu}, \tag{5.17}$$

and from constraint (5.5) for $t = I_i^\alpha$ we get, $C_i^\alpha \leq \frac{1}{\alpha}\tau_{I_i^\alpha}$.

Let $\bar{C}_i = \sum_{j=1}^{q}\sum_{t=1}^{T}\tau_{t-1}\bar{x}_{ijt}$ denote the optimal fractional completion time given by the optimal solution of the relaxed linear program (5.3)-(5.6). Since it is possible that

$\sum_{j=1}^{q}\sum_{t=1}^{I_i^\alpha}\bar{x}_{ijt} > \alpha$; we define $X_i^{(1)} = \alpha - \sum_{j=1}^{q}\sum_{t=1}^{I_i^\alpha-1}\bar{x}_{ijt}$ and $X_i^{(2)} = \sum_{j=1}^{q}\sum_{t=1}^{I_i^\alpha}\bar{x}_{ijt} - \alpha$, thus $X_i^{(1)} + X_i^{(2)} = \sum_{j=1}^{q}\bar{x}_{ijI_i^\alpha}$, and we can rewrite

$$\bar{C}_i = \sum_{j=1}^{q}\sum_{t=1}^{I_i^\alpha-1}\tau_{t-1}\bar{x}_{ijt} + \tau_{I_i^\alpha-1}X_i^{(1)} + \tau_{I_i^\alpha-1}X_i^{(2)} + \sum_{j=1}^{q}\sum_{t=I_i^\alpha+1}^{T}\tau_{t-1}\bar{x}_{ijt}, \tag{5.18}$$

and eliminating the lower terms of the previous sum we get that,

$$\begin{aligned}\bar{C}_i &\geq \tau_{I_i^\alpha-1}X_i^{(2)} + \sum_{j=1}^{q}\sum_{t=I_i^\alpha+1}^{T}\tau_{t-1}\bar{x}_{ijt} \\ &\geq \tau_{I_i^\alpha-1}X_i^{(2)} + \sum_{j=1}^{q}\sum_{t=I_i^\alpha+1}^{T}\tau_{I_i^\alpha-1}\bar{x}_{ijt} = \tau_{I_i^\alpha-1}(1-\alpha).\end{aligned} \tag{5.19}$$

Because $\tau_{I_i^\alpha} = (1+\epsilon)\tau_{I_i^\alpha-1}$, from (5.17) and (5.20) we get that $C_i^\alpha \leq \frac{(1+\epsilon)}{\alpha(1-\alpha)}\bar{C}_i \Rightarrow \sum_{i=1}^{n} w_i C_i^\alpha \leq \frac{(1+\epsilon)}{\alpha(1-\alpha)}\sum_{i=1}^{n} w_i\bar{C}_i$. From this, (5.14) and (5.16) it follows that,

$$\sum_{i=1}^{n}\mathcal{R}_i(\boldsymbol{\psi}^{(i)\alpha}) + \sum_{i=1}^{n} w_i C_i^\alpha \leq \frac{(1+\epsilon)}{\alpha(1-\alpha)}\left[\sum_{i=1}^{n}\sum_{j=1}^{q}\sum_{t=1}^{T}\mathcal{R}_i(\boldsymbol{\Psi}^{(j)})x_{ijt}^* + \sum_{i=1}^{n} w_i C_i^*\right], \tag{5.20}$$

and we set $\alpha = \arg\min_{0\leq\alpha\leq1}\left\{\frac{1}{\alpha(1-\alpha)}\right\} = \frac{1}{2}$, to minimize the bound, and get the desired approximation ratio. □

5.2.2 Single machine problem with precedence and release date constraints.

We now analyse the case with precedence constraints and release dates. Release dates makes the problem somewhat harder since they can introduce idle times between jobs.

Theorem 5.3. *The SAIAS algorithm with $\alpha = \sqrt{2}-1$ is a $(3+2\sqrt{2}+\epsilon)$-approximation algorithm for the $1|r_i, prec|\sum\mathcal{R}_i(\boldsymbol{\psi}^{(i)}) + w_iC_i$ problem, with a general non-negative $\mathcal{R}_i(\boldsymbol{\psi})$ resource cost function.*

Proof. The bounds for the resource cost terms computed in equation (5.15) are still valid when there is idle time between jobs, and we have that,

$$\mathcal{R}_i(\boldsymbol{\psi}^{(i)\alpha}) \leq \frac{(1+\epsilon)}{\alpha(1-\alpha)}\sum_{j=1}^{q}\sum_{u=1}^{T}\mathcal{R}_i(\boldsymbol{\Psi}^{(j)})\bar{x}_{iju} \leq \frac{(1+\epsilon)(1+\alpha)}{\alpha(1-\alpha)}\sum_{j=1}^{q}\sum_{u=1}^{T}\mathcal{R}_i(\boldsymbol{\Psi}^{(j)})\bar{x}_{iju}. \tag{5.21}$$

When bounding the completion time C_i^α, given the sorting done in step 5 of the SAIAS algorithm, now one has to consider all the jobs up to the ones in set $J_{I_i^\alpha}$, and thus,

$$C_i^\alpha \leq \max_{j\in\{J_1,\ldots,J_{I_i^\alpha}\}} r_j + \sum_{j\in\{J_1,\ldots,J_{I_i^\alpha}\}} \frac{\rho_j}{s_j^\alpha}. \tag{5.22}$$

Since all jobs that have been at least partially processed up to time interval I_t need to be released before τ_t, it follows that $\max_{j\in\{J_1,\ldots,J_{I_i^\alpha}\}} r_j \leq \tau_{I_i^\alpha}$. On the other hand, we also have that,

$$\sum_{j\in\{J_1,\ldots,J_{I_i^\alpha}\}} \frac{\rho_j}{s_j^\alpha} = \frac{1}{\alpha} \sum_{j\in\{J_1,\ldots,J_{I_i^\alpha}\}} \sum_{l=1}^{q} \sum_{u=1}^{I_j^\alpha} \frac{\rho_j}{\sigma_l} \tilde{x}_{jlu} \leq \frac{1}{\alpha} \sum_{j=1}^{n} \sum_{l=1}^{q} \sum_{u=1}^{I_i^\alpha} \frac{\rho_j}{\sigma_l} \bar{x}_{jlu} \leq \frac{1}{\alpha} \tau_{I_i^\alpha}, \tag{5.23}$$

where the last inequality follows from constraint (5.5) with $t = I_i^\alpha$. Thus, $C_i^\alpha \leq \frac{(1+\alpha)}{\alpha}\tau_{I_i^\alpha}$. Since $\bar{C}_i = \sum_{j=1}^{q}\sum_{t=1}^{T} \tau_{t-1}\bar{x}_{ijt}$, (5.20) is still valid and because $\tau_{I_i^\alpha} = (1+\epsilon)\tau_{I_i^\alpha - 1}$, we get,

$$C_i^\alpha \leq \frac{(1+\epsilon)(1+\alpha)}{\alpha(1-\alpha)} \bar{C}_i \Rightarrow \sum_{i=1}^{n} w_i C_i^\alpha \leq \frac{(1+\epsilon)(1+\alpha)}{\alpha(1-\alpha)} \sum_{i=1}^{n} w_i \bar{C}_i. \tag{5.24}$$

Finally, from (5.21) and (5.24) it follows that,

$$\sum_{i=1}^{n} \mathcal{R}_i(\boldsymbol{\psi}^{(i)\alpha}) + \sum_{i=1}^{n} w_i C_i^\alpha \leq \frac{(1+\epsilon)(1+\alpha)}{\alpha(1-\alpha)} \left[\sum_{i=1}^{n}\sum_{j=1}^{q}\sum_{t=1}^{T} \mathcal{R}_i(\boldsymbol{\Psi}^{(j)}) x_{ijt}^* + \sum_{i=1}^{n} w_i C_i^* \right], \tag{5.25}$$

and by setting $\alpha = \arg\min_{0\leq\alpha\leq 1} \left\{ \frac{(1+\alpha)}{\alpha(1-\alpha)} \right\} = \sqrt{2} - 1$, we get the required approximation ratio. □

Chapter 6

Scheduling with Uncertain Job Sizes

A basic assumption in our previous chapters, even in the online setting, was that we learn the size of a job at least as soon as the job arrives. In the CPU scheduling setting there are many cases in which this is not doable and there is no information or it is not possible to deduce the size of a job until it is actually finished [Kalyanasundaram and Pruhs, 2000]. This setting in which no information of job sizes is known or deduced when jobs are released is known as non-clairvoyant scheduling and there are some approximation algorithms and heuristics for certain scheduling metrics. See [Kalyanasundaram and Pruhs, 2000] and its references for further details. To the best of our knowledge, there are no results for the non-clairvoyant energy aware scheduling problem, that is when we are not only interested in optimizing a scheduling metric, but also minimizing some energy consumption cost.

In this chapter we explore this setting, where jobs are processed in a single machine with preemption, and we are interested in minimizing the sum of the total weighed completion time and the energy cost of processing the jobs. Unlike the previous chapters, the machine does not know nor it can deduce the exact size of the jobs until it finishes them, but we assume that job sizes have a known discrete probability distribution and that all jobs share the same size distribution (i.e. they are i.i.d).

6.1 Problem Formulation

The problem setting is as follows: we are given n jobs, where each job $i \in \mathbf{N} = \{1, \ldots, n\}$ has an unknown processing requirement of $\rho_i \in \Psi = \{\psi_1, \ldots, \psi_q\}$ machine cycles, with $\psi_0 = 0$ and $\mathbb{P}(\rho_i = \psi_k) = f_k$. For convenience let $\Delta\psi_k = \psi_{k+1} - \psi_k > 0$, which is the minimum additional processing requirements if after ψ_k cycles the job has not finished. All jobs run on the same machine, which at any time runs at one of $m+1$ possible speeds $\mathbf{S} = \{\sigma_0, \sigma_1, \ldots, \sigma_m\}$, with $\sigma_0 = 0$. Preemption is allowed and we will assume all jobs are available at time $t = 0$ (i.e. $r_i = 0$, $\forall i$).

Let C_i denote the final completion time of job i, then the schedule performance cost of job i will be given by $h_i(C_i)$, where $h_i(C)$ is a monotonically increasing linear function (i.e. $h_i(C_i + \delta) = h_i(C_i) + h_i(\delta)$). Additionally, if at time t job i runs at speed $s_i(t)$, then let $\mathcal{P}_i(s_i(t))$ denote the total energy cost rate of job i (i.e. the power consumption), that is, the cost per unit of time at time t. We assume $\mathcal{P}_i(s)$ is convex.

Let $\mathbf{s} = \begin{bmatrix} s_1(t) & \ldots & s_n(t) \end{bmatrix}'$ denote the vector of speed functions for each job. Clearly, for $\mathbf{s}$ to be a feasible vector we need that if for some job i and time t, $s_i(t) > 0$, then $s_j(t) = 0$, $\forall j \neq i$.

We are interested in minimizing the sum of the total scheduling performance and total energy cost for all jobs, that is:

$$F(\mathbf{s}) = \mathbb{E}\left[\sum_{i=1}^{n} \int_0^T \mathcal{P}_i(s_i(t))dt + \sum_{i=1}^{n} h_i(C_i)\right]. \tag{6.1}$$

6.2 Dynamic Programming Model

We propose solving this problem using dynamic programming. At any time step, let $\mathbf{J}$ denote the jobs that still require processing and let $\boldsymbol{\Xi}$ denote the amount of work each job has received by that time. We now use $(\mathbf{J}, \boldsymbol{\Xi})$ to denote the state space for our problem.

We prove the following lemma:

Lemma 6.1. *Given a state $(\mathbf{J}, \boldsymbol{\Xi})$, if it is optimal to process the unfinished job i next and has*

already received ψ_k processing cycles, then there exists an optimal solution where we process job i next for at least $\Delta\psi_k$ cycles at a constant speed.

Proof. We prove this lemma by contradiction. Let job i be the optimal job to process at time t_1, and let $(\mathbf{J}, \boldsymbol{\Xi})$ be the state of the system at that time, with $\xi_i = \psi_k$. Let $\delta_1 < \Delta\psi_k$ denote the optimal number of processing cycles assigned to job i until it is preempted and $s_i(t)$, $t \in [t_1, t_2]$ the optimal speed at which the δ_i cycles are processed, i.e. $\int_{t_1}^{t_2} s_i(t)dt = \delta_i$.

Because the energy cost is convex and there are no jobs arriving while job i is being processed, the speed to process the δ_1 cycles will be constant, i.e. $s_i(t) \equiv \sigma_j$, for some j and $\forall t \in [t_1, t_2]$. Thus, we will process job i for $\frac{\delta_1}{\sigma_j}$ time units after which we switch to a different job. Note that job i is not finished at this point, since its length is at least ψ_{k+1}, so there must be at least one other time t_3 at which another $\delta_2 \geq \Delta\psi_k - \delta_1 > 0$ cycles of job j are processed.

Let $\mathbf{I}$ denote the set of jobs that finish in the interval $[t_1, t_3]$ in this optimal solution. Now consider a different solution in which the δ_1 cycles of job i are processed at the end of the interval (from $t_3 - \frac{\delta_1}{\sigma_j}$ to t_3), and because there are no release dates all other job pieces in $[t_1, t_3]$, are processed $\frac{\delta_1}{\sigma_j}$ time units earlier. Because $i \notin \mathbf{I}$ the completion time of i does not change. Additionally, because we are only moving job pieces the speeds don't change and thus the total energy cost remains the same.

If $\mathbf{I} \neq \emptyset$ then all jobs in $\mathbf{I}$ finish earlier and because the energy costs and the completion time of job i don't change, the new solution is smaller than the optimal one by $\sum_{j \in \mathbf{I}} w_j \frac{\delta_1}{\sigma_j} > 0$, which is a contradiction. On the other hand, if $\mathbf{I} = \emptyset$, then moving the piece of job i to the end of the interval $[t_1, t_3]$ has the same total cost. Hence, the new solution is also optimal but now we process $\delta_1 + \delta_2$ cycles of job i together. If $\delta_1 + \delta_2 \geq \Delta\psi_k$ we have found an optimal solution where at least $\Delta\psi_k$ cycles are processed, otherwise we can repeat the previous argument to keep on joining pieces of job i until at least $\Delta\psi_k$ cycles are processed. □

Because of Lemma 6.1 we can assume, w.l.o.g., that if the optimal solution starts processing a job with ψ_k cycles already processed, it will process at least $\Delta\psi_k$ units before it could preempt to another job. Also note that using the same argument from Lemma 6.1 we can

also prove that the optimal solution will not process more than $\sum_{j=k}^{l} \Delta\psi_j$ cycles, for some $l \leq q-1$, and thus we will only preempt when the work state $\mathbf{\Xi}$ is such that $\xi_i = \psi_{k_i}$. Because of this result and to simplify the notation, $\mathbf{\Xi}$ will still denote the amount of work each job has received but now if job i has been processed for ψ_k cycles, then $\xi_i = k$, and thus $\mathbf{\Xi} = \begin{bmatrix} k_1 & \dots & k_n \end{bmatrix}'$.

Let $V(\mathbf{J}, \mathbf{\Xi})$ denote the optimal cost when only jobs in $\mathbf{J}$ are available and ψ_{ξ_i} cycles have been processed of each job i. If at this point it is optimal to process an unfinished job i at speed σ_j, then the expected cost for the remaining processing requirements is given by

$$V(\mathbf{J}, \mathbf{\Xi} + e_i)\frac{\bar{F}_{\xi_i+1}}{\bar{F}_{\xi_i}} + V(\mathbf{J}\backslash\{i\}, \mathbf{\Xi} + e_i)\frac{f_{\xi_i}}{\bar{F}_{\xi_i}} + \frac{\Delta\psi_{\xi_i}}{\sigma_j}\mathcal{P}_i(\sigma_j) + \sum_{l\in\mathbf{J}} h_l\left(\frac{\Delta\psi_{\xi_i}}{\sigma_j}\right), \tag{6.2}$$

where F_k is the cdf of the size distribution of the jobs. The first element in (6.2) represents the cost if the job is longer than ψ_{ξ_i+1} cycles, the second term is the cost if job i is finished, and the last two terms are the additional energy cost of job i and the additional delay incurred by all jobs that have not been finished yet.

Using (6.2) we get the following recursion which we can use to find the optimal policy

$$V(\mathbf{J}, \mathbf{\Xi}) = \min_{i,j}\left\{V(\mathbf{J}, \mathbf{\Xi} + e_i)\frac{\bar{F}_{\xi_i+1}}{\bar{F}_{\xi_i}} + V(\mathbf{J}\backslash\{i\}, \mathbf{\Xi} + e_i)\frac{f_{\xi_i}}{\bar{F}_{\xi_i}} + \frac{\Delta\psi_{\xi_i}}{\sigma_j}\mathcal{P}_i(\sigma_j) + \sum_{l\in\mathbf{J}} h_l\left(\frac{\Delta\psi_{\xi_l}}{\sigma_j}\right)\right\} \tag{6.3}$$

and we are interested in computing $\min_{\mathbf{s}} F(\mathbf{s}) = V(\mathbf{N}, \mathbf{0})$. The size of the state space increases exponentially with the number of jobs, so using this recursion is not practical for even medium sized instances (e.g. 10 jobs and 15 possible sizes). Since it is not practical, we will first simplify the problem slightly and then propose a policy that returns the same result as solving the recursion in (6.3).

6.3 Weighted Completion Time and Polynomial Power Cost

We will assume that $\mathcal{P}_i(s) = v_i s^\beta$, with $\beta > 2$, thus the energy cost if $\Delta\psi_k$ cycles of job i are processed at speed s_i is given by $E_i(s_i) = v_i \Delta\psi_k s_i^{\beta-1}$. Additionally, let $h_i(C_i) = w_i C_i$, i.e weighted completion time. Finally, we assume that $\mathbf{S} = \mathbb{R}_+^n$. Let $W_{\mathbf{J}} = \sum_{i\in\mathbf{J}} w_i$, then the recursion in (6.3) is simplified to

$$V(\mathbf{J}, \boldsymbol{\Xi}) = \min_{i,\mathbf{s}} \left\{ V(\mathbf{J}, \boldsymbol{\Xi} + e_i)\frac{\bar{F}_{\xi_i+1}}{\bar{F}_{\xi_i}} + V(\mathbf{J}\backslash\{i\}, \boldsymbol{\Xi} + e_i)\frac{f_{\xi_i}}{\bar{F}_{\xi_i}} + \Delta\psi_{\xi_i} D_{\mathbf{J}}(i, s_i) \right\}, \tag{6.4}$$

where $D_{\mathbf{J}}(i, s_i) = v_i s_i^{\beta-1} + \frac{W_{\mathbf{J}}}{s_i}$. For any given i the first two terms in (6.4) are constant, thus, we only need to minimize $D_{\mathbf{J}}(i, s_i)$. Using the results from Chapter 4 it is easy to compute the optimal value of $D_{\mathbf{J}}(i, s_i)$, which is achieved at

$$s_i^* = \sqrt[\beta]{\frac{W_{\mathbf{J}}}{(\beta-1)v_i}}, \tag{6.5}$$

and thus, by defining $\mathcal{B} \equiv \frac{\beta}{(\beta-1)^b}$ and $b \equiv \frac{\beta-1}{\beta}$, we get that $D_{\mathbf{J}}(i, s_i^*) = \mathcal{B}\sqrt[\beta]{v_i W_{\mathbf{J}}^{\beta-1}}$. Hence, (6.4) is further simplified to

$$V(\mathbf{J}, \boldsymbol{\Xi}) = \min_{i} \left\{ V(\mathbf{J}, \boldsymbol{\Xi} + e_i)\frac{\bar{F}_{\xi_i+1}}{\bar{F}_{\xi_i}} + V(\mathbf{J}\backslash\{i\}, \boldsymbol{\Xi} + e_i)\frac{f_{\xi_i}}{\bar{F}_{\xi_i}} + \Delta\psi_{\xi_i}\mathcal{B}\sqrt[\beta]{v_i W_{\mathbf{J}}^{\beta-1}} \right\}. \tag{6.6}$$

Now we can compute $V(\mathbf{N}, \mathbf{0})$ by starting from $V(i, q-1) = \Delta\psi_{q-1} B\sqrt[\beta]{v_i w_i^{\beta-1}}$, for all i, and then computing the other terms backwards, noting that

$$V(i, k) = V(i, k+1)\frac{\bar{F}_{k+1}}{\bar{F}_k} + \Delta\psi_k \mathcal{B}\sqrt[\beta]{v_i w_i^{\beta-1}}.$$

6.3.1 Completion Time and Job Independent Power Cost

We now consider a further simplification: let $w_i = w$ and $v_i = v$ $\forall i$. Since in this case two jobs that have received the same amount of work are indistinguishable, we can use a simpler state description. Let $\boldsymbol{\Theta} = \left[\theta_1, \ldots, \theta_q\right]$ denote the new state space, where θ_k denotes the number of

jobs that we currently know have at least ψ_k cycles as work requirement. We can think of $\mathbf{\Theta}$ representing q different queues the machine must serve, and once it chooses to process a job from queue j then the job either leaves (i.e. θ_j is reduced by 1) or it moves to the following queue (i.e. θ_j is reduced by 1 and θ_{j+1} is increased by 1). For example, in the beginning we have $\theta_1 = n$ and $\theta_k = 0$ for the rest since all jobs will have at least ψ_1 but we know nothing more. Once we process one job for $\Delta\psi_0$ cycles, we have that $\theta_1 = n - 1$ and $\theta_2 = 1$ if the selected job is not finished. For notation simplicity we will denote $\mid \mathbf{\Theta} \mid = \sum_{k=1}^{q} \theta_k$, the number of jobs that are still available.

Lemma 6.1 is still valid under this state description so given a current state $\mathbf{\Theta}$ then the optimal cost for the remaining jobs will be given by

$$V(\mathbf{\Theta}) = \min_k \left\{ V(\mathbf{\Theta} - e_k + e_{k+1}) \frac{\bar{F}_{k+1}}{\bar{F}_k} + V(\mathbf{\Theta} - e_k) \frac{f_k}{\bar{F}_k} + \Delta\psi_{k-1} D \sqrt[\beta]{\mid \mathbf{\Theta} \mid^{\beta-1}} \right\}, \tag{6.7}$$

where $D = \mathcal{B} \sqrt[\beta]{v w^{\beta-1}}$.

6.3.1.1 Two Queue System ($q = 2$)

For the case when $\Psi = \{\psi_1, \psi_2\}$, with $f = \begin{bmatrix} p & 1-p \end{bmatrix}$, the DP recursion (6.7) can be written as

$$\begin{aligned} V(\theta_1, \theta_2) = \min \big\{ & V(\theta_1 - 1, \theta_2 + 1)(1-p) + V(\theta_1 - 1, \theta_2)\, p + \Delta\psi_0 D(\theta_1 + \theta_2)^b, \\ & V(\theta_1, \theta_2 - 1) + \Delta\psi_1 D(\theta_1 + \theta_2)^b \big\}, \end{aligned} \tag{6.8}$$

where $b = \frac{\beta-1}{\beta}$.

From (6.8) note that whenever we are in a state where $\theta_1 = 0$, the only available decision is to process a job from θ_2 and thus we get that $\forall \xi > 0$,

$$V(0, \xi) = V(0, \xi - 1) + \Delta\psi_1 D \xi^b = \Delta\psi_1 D \sum_{i=1}^{\xi} i^b. \tag{6.9}$$

Similarly, whenever $\theta_2 = 0$ we can only process a job from θ_1 and thus,

$$V(\xi, 0) = V(\xi - 1, 0)\, p + V(\xi - 1, 1)(1-p) + \Delta\psi_0 D \xi^b \tag{6.10}$$

but we cannot further reduce it since we don't know what decision will be made at $V(\xi-1,1)$.

We now prove the following theorem:

Theorem 6.1. *Given $\mathbf{\Theta}=(\tau,n-\tau)$, with $0<\tau<n$ for any $n\geq 2$, then if $\frac{\Delta\psi_0}{\Delta\psi_1}\geq p$ it is optimal to process a job from θ_2, whereas if $\frac{\Delta\psi_0}{\Delta\psi_1}\leq p$ it is optimal to process a job from θ_1.*

Proof. We prove it by double induction on the total number of jobs on each queue (θ_1 or θ_2). First, for $n=2$, we only have $\tau=1$, and from (6.8) we get that

$$V(1,1)=\min\left\{V(0,1)\,p+V(0,2)\,(1-p)+\Delta\psi_0 D2^b, V(1,0)+\Delta\psi_1 D2^b\right\}. \tag{6.11}$$

By using (6.9) and (6.10) in (6.11) we get that

$$V(1,1)=D\min\left\{\Delta\psi_0\left[2^b\right]+\Delta\psi_1\left[-p2^b+1+2^b\right],\Delta\psi_0+\Delta\psi_1\left[1-p+2^b\right]\right\}. \tag{6.12}$$

Note that D, and thus v and w, do not affect the decision. From (6.12) we conclude that it will be optimal to process a job from θ_2 if

$$\Delta\psi_0\left[2^b\right]+\Delta\psi_1\left[-p2^b+1+2^b\right]\geq\Delta\psi_0+\Delta\psi_1\left[1-p+2^b\right]\Leftrightarrow\frac{\Delta\psi_0}{\Delta\psi_1}\geq p, \tag{6.13}$$

and similarly, it will be optimal to process a job from θ_1 if $\frac{\Delta\psi_0}{\Delta\psi_1}\leq p$.

We now continue with the induction proof for the two cases, $\frac{\Delta\psi_0}{\Delta\psi_1}\geq p$ and $\frac{\Delta\psi_0}{\Delta\psi_1}\leq p$, separately. First let $\frac{\Delta\psi_0}{\Delta\psi_1}\geq p$, i.e. we assume that for $0<\tau<\xi$, and $\xi\leq n$, at state $\mathbf{\Theta}=(\tau,\xi-\tau)$ it will be optimal to process a job from θ_2. For this case note that (6.10) does have a close form, since $V(\xi-1,1)=V(\xi-1,0)+\Delta\psi_1 D\xi^b$, and thus

$$V(\xi,0)=\Delta\psi_0\left[\sum_{i=1}^{\xi}i^b\right]+\Delta\psi_1\left[(1-p)\sum_{i=1}^{\xi}i^b\right], \tag{6.14}$$

so the hypothesis is that for $0<\tau<\xi$, and $\xi\leq n$,

$$V(\tau,\xi-\tau)=\Delta\psi_0\left[\sum_{i=1}^{\tau}i^b\right]+\Delta\psi_1\left[\sum_{i=1}^{\xi}i^b-p\sum_{i=1}^{\tau}i^b\right]. \tag{6.15}$$

We now prove that it is optimal to process a job from θ_2 for $n+1$, which is done by induction on τ. First, for $\tau = 1$ we get that

$$V(0,n)\,p + V(0,n+1)\,(1-p) + \Delta\psi_0(n+1)^b = \Delta\psi_0(n+1)^b + \Delta\psi_1\left[\sum_{i=1}^{n+1} i^b - p(n+1)^b\right],$$

and

$$V(1,n-1) + \Delta\psi_1(n+1)^b = \Delta\psi_0 + \Delta\psi_1\left[\sum_{i=1}^{n+1} i^b - p\right],$$

thus,

$$\Delta\psi_0(n+1)^b + \Delta\psi_1\left[\sum_{i=1}^{n+1} i^b - p(n+1)^b\right] \geq \Delta\psi_0 + \Delta\psi_1\left[\sum_{i=1}^{n+1} i^b - p\right] \Leftrightarrow \frac{\Delta\psi_0}{\Delta\psi_1} \geq p. \tag{6.16}$$

We now assume it is optimal for $\mathbf{\Theta} = (\theta_1, n+1-\theta_1)$, with $\theta_1 \leq \tau$, and prove it for $\theta_1 = \tau + 1$. In this case

$$\begin{aligned} &V(\tau,n-\tau)\,p + V(\tau,n+1-\tau)\,(1-p) + \Delta\psi_0(n+1)^b \\ &= \Delta\psi_0\left[\sum_{i=1}^{\tau} i^b + (n+1)^b\right] + \Delta\psi_1\left[\sum_{i=1}^{n+1} i^b - p\sum_{i=1}^{\tau} i^b - p(n+1)^b\right], \end{aligned}$$

and

$$V(\tau+1,n-1-\tau) + \Delta\psi_1(n+1)^b = \Delta\psi_0\left[\sum_{i=1}^{\tau+1} i^b\right] + \Delta\psi_1\left[\sum_{i=1}^{n+1} i^b - p\sum_{i=1}^{\tau+1} i^b\right]$$

thus,

$$\begin{aligned} \frac{\Delta\psi_0}{\Delta\psi_1}\left[\sum_{i=1}^{\tau} i^b + (n+1)^b\right] + \sum_{i=1}^{n+1} i^b - p\sum_{i=1}^{\tau} i^b - p(n+1)^b \geq \frac{\Delta\psi_0}{\Delta\psi_1}\left[\sum_{i=1}^{\tau+1} i^b\right] + \sum_{i=1}^{n+1} i^b - p\sum_{i=1}^{\tau+1} i^b, \\ \Leftrightarrow \frac{\Delta\psi_0}{\Delta\psi_1} \geq p. \end{aligned} \tag{6.17}$$

Hence, if $\frac{\Delta\psi_0}{\Delta\psi_1} \geq p$ for any $0 < \tau < n$, with $n \geq 2$, at state $\mathbf{\Theta} = (\tau, n-\tau)$ it is optimal to process a job from θ_2.

For the case $\frac{\Delta\psi_0}{\Delta\psi_1} \leq p$, we have proved that for $n = 2$ it is optimal to process a job from θ_1. We assume it is optimal to process a job from θ_1 for every state $\mathbf{\Theta} = (\tau, \xi - \tau)$ with $0 \leq \tau < \xi$, and $\xi \leq n$, i.e.

$$\begin{aligned} V(\tau, \xi-\tau) =& V(\tau-1, \xi-\tau)\, p + V(\tau-1, \xi-\tau+1)\,(1-p) + \Delta\psi_0 \xi^b, \\ =& \sum_{i=0}^{\tau} \binom{\tau}{i} p^{\tau-i}(1-p)^i V(0, \xi-\tau+i) + \Delta\psi_0 \sum_{j=0}^{\tau-1}\sum_{i=0}^{j} \binom{j}{i} p^{j-i}(1-p)^i (\xi-j+1)^b. \end{aligned} \tag{6.18}$$

Now we prove it for $n+1$ by using induction on τ. For $\tau = 1$ we have that,

$$V(0,n)\,p + V(0,n+1)\,(1-p) + \Delta\psi_0(n+1)^b = \Delta\psi_0(n+1)^b + \Delta\psi_1\left[\sum_{i=1}^{n+1} i^b - p(n+1)^b\right],$$

and

$$V(1,n-1) + \Delta\psi_1(n+1)^b = \Delta\psi_0 n^b + \Delta\psi_1\left[\sum_{i=0}^{n} i^b + (n+1)^b - pn^b\right],$$

thus,

$$\Delta\psi_0(n+1)^b + \Delta\psi_1\left[\sum_{i=1}^{n+1} i^b - p(n+1)^b\right] \leq \Delta\psi_0 n^b + \Delta\psi_1\left[\sum_{i=0}^{n} i^b + (n+1)^b - pn^b\right] \Leftrightarrow \frac{\Delta\psi_0}{\Delta\psi_1} \leq p. \tag{6.19}$$

We next assume it is optimal to process a job from θ_1 for $\mathbf{\Theta} = (\theta_1, n+1-\theta_1)$, with $\theta_1 \leq \tau$, and finally we prove it for $\theta_1 = \tau + 1$, i.e. we need to prove that

$$V(\tau, n-\tau)\,p + V(\tau, n-\tau+1)\,(1-p) + \Delta\psi_0(n+1)^b \leq V(\tau+1, n-\tau-1) + \Delta\psi_1(n+1)^b. \tag{6.20}$$

From (6.18) we get that the LHS of (6.20) can be rewritten as

$$\sum_{i=0}^{\tau+1} \binom{\tau+1}{i} p^{\tau+1-i}(1-p)^i V(0, n-\tau+i) + \Delta\psi_0 \sum_{j=0}^{\tau}\sum_{i=0}^{j} \binom{j}{i} p^{j-i}(1-p)^i (n+1-j+i)^b. \tag{6.21}$$

Similarly, the RHS of (6.20) can be rewritten as

$$\sum_{i=0}^{\tau+1}\binom{\tau+1}{i}p^{\tau+1-i}(1-p)^i V\left(0,n-\tau-1+i\right)+\Delta\psi_0\sum_{j=0}^{\tau}\sum_{i=0}^{j}\binom{j}{i}p^{j-i}(1-p)^i(n-j+i)^b+\Delta\psi_1(n+1)^b, \tag{6.22}$$

and by noting that $V\left(0,n-\tau+i\right)=V\left(0,n-\tau-1+i\right)+\Delta\psi_1(n-\tau+i)^b$, we get that proving (6.20) is equivalent to proving that

$$\begin{aligned}\Delta\psi_1\sum_{i=0}^{\tau+1}\binom{\tau+1}{i}p^{\tau+1-i}(1-p)^i(n-\tau+i)^b+\Delta\psi_0\sum_{j=0}^{\tau}\sum_{i=0}^{j}\binom{j}{i}p^{j-i}(1-p)^i(n+1-j+i)^b\\ \leq\Delta\psi_0\sum_{j=0}^{\tau}\sum_{i=0}^{j}\binom{j}{i}p^{j-i}(1-p)^i(n-j+i)^b+\Delta\psi_1(n+1)^b,\end{aligned} \tag{6.23}$$

or equivalently,

$$\begin{aligned}\frac{\Delta\psi_0}{\Delta\psi_1}\left[\sum_{j=0}^{\tau}\sum_{i=0}^{j}\binom{j}{i}p^{j-i}(1-p)^i(n+1-j+i)^b-\sum_{j=0}^{\tau}\sum_{i=0}^{j}\binom{j}{i}p^{j-i}(1-p)^i(n-j+i)^b\right]\\ \leq-\sum_{i=0}^{\tau+1}\binom{\tau+1}{i}p^{\tau+1-i}(1-p)^i(n-\tau+i)^b+(n+1)^b.\end{aligned} \tag{6.24}$$

We can write the RHS of (6.24) as,

$$-\sum_{i=0}^{\tau}\binom{\tau+1}{i}p^{\tau+1-i}(1-p)^i(n-\tau+i)^b-(1-p)^{\tau+1}(n+1)^b+(n+1)^b,$$

and by expanding the binomial $(1-p)^{\tau+1}$, changing the sum index, and reordering, we get that the RHS of (6.24) is equal to

$$p\left[(n+1)^b\left\{\sum_{i=0}^{\tau}\binom{\tau+1}{i}(-p)^{\tau-i}\right\}+\sum_{i=1-\tau}^{0}(n+i)^b\left\{-\binom{\tau+1}{i+\tau}p^{-i}(1-p)^{i+\tau}\right\}-(n-\tau)^b p^\tau\right]. \tag{6.25}$$

Similarly, by exchanging the sums, changing the sum indices, and regrouping powers, we get that the LHS of (6.24) can be rewritten as,

$$\frac{\Delta\psi_0}{\Delta\psi_1}\left[(n+1)^b\left\{\sum_{i=0}^{\tau}(1-p)^i\right\}+\sum_{i=1-\tau}^{0}(n+i)^b p^{-i}\left\{(1-p)^{i+j-1}\sum_{j=1-i}^{\tau}\left[\binom{j+1}{i+j}p-\binom{j}{i+j}\right]-1\right\}\right.$$
$$\left.-(n-\tau)^b p^{\tau}\right]. \tag{6.26}$$

Finally, we can prove that

$$\sum_{i=0}^{\tau}\binom{\tau+1}{i}(-p)^{\tau-i}=\sum_{i=0}^{\tau}(1-p)^i,$$

by noting that

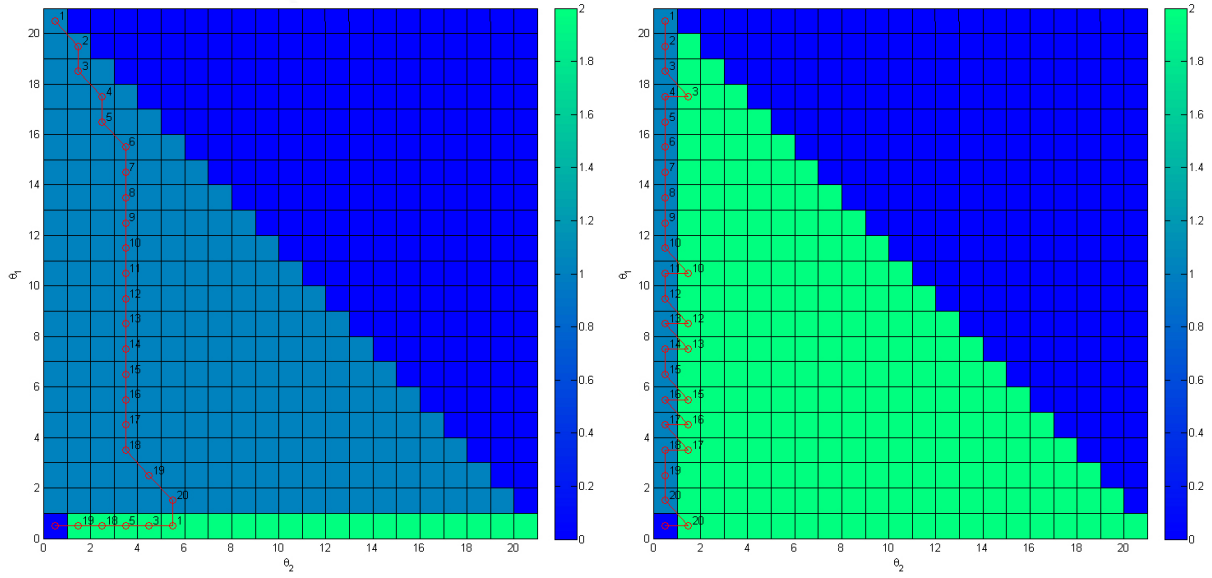
$$\sum_{i=\tau-k}^{\tau}\binom{i}{\tau-k}=\binom{\tau+1}{k},$$

and similarly we can prove that

$$-\binom{\tau+1}{i+\tau}p^{-i}(1-p)^{i+\tau}=p^{-i}\left\{(1-p)^{i+j-1}\sum_{j=1-i}^{\tau}\left[\binom{j+1}{i+j}p-\binom{j}{i+j}\right]-1\right\},$$

for $i=1-\tau,\ldots,0$, and thus, all the coefficients of the terms $(n+i)^b$, for $i=\{-\tau,\ldots,1\}$ are the same in (6.25) and (6.26). Hence, we get that (6.20) is true if $\frac{\Delta\psi_0}{\Delta\psi_1}\leq p$, completing the proof. □

Using Theorem 6.1 we obtain an optimal policy without computing $V(\boldsymbol{\Theta})$ for all possible states $\boldsymbol{\Theta}$. Some implications Theorem 6.1 are very intuitive. If the probability p of jobs being small is large, or the work required in the first queue is much smaller than the one in the second queue, it is optimal, in expectation, to always process jobs from the first stage. On the other if such probability is small (i.e. the probability of jobs being large is big) or the work in the second queue is much smaller, then it is an optimal policy to process jobs all the way until they finish.



(a) Instance with $\frac{\Delta\psi_0}{\Delta\psi_1} < p$

(b) Instance with $\frac{\Delta\psi_0}{\Delta\psi_1} > p$

Figure 6.1: Example Instances for $q = 2$

It is important to note that the optimal policy only depends only on the values of $\Delta\psi_0$, $\Delta\psi_1$, and p, and not the amount of jobs on each state, nor the number of jobs n or the weights v and w. Also, if $\frac{\Delta\psi_0}{\Delta\psi_1} = p$ any policy would be optimal.

Figure 6.1 shows the optimal policies for sample instances of the two possible cases. In Figure 6.1a we have that $\frac{\Delta\psi_0}{\Delta\psi_1} < p$, in which, by Theorem 6.1, it is optimal to process jobs from stage θ_1. Thus, when a job is not finished the policy chooses new job from θ_1 instead of finishing the job completely. The other case is shown in Figure 6.1b. In this case, whenever a job from θ_1 is processed and not finished, the optimal policy is then to finish it before processing a new job from θ_1. The colors in each square in both figures 6.1a and 6.1b indicate the optimal policy if that specific state is achieved.

6.3.1.2 Multiple State System

Recall equation (6.7) which determines the optimal cost given a current state $\mathbf{\Theta}$.

$$V(\mathbf{\Theta}) = \min_k \left\{ V(\mathbf{\Theta} - e_k + e_{k+1}) \frac{\bar{F}_{k+1}}{\bar{F}_k} + V(\mathbf{\Theta} - e_k) \frac{f_k}{\bar{F}_k} + \Delta\psi_{k-1} D \sqrt[\beta]{\mid \mathbf{\Theta} \mid^{\beta-1}} \right\}.$$

Given the number of states q and the number of jobs n in an instance, we need $O(q^n)$ computations to find the optimal rule, i.e. which k to choose at each state $\boldsymbol{\Theta}$, making it non-practical for any reasonable applications.

Let $\boldsymbol{\eta}(\boldsymbol{\Theta}) = \begin{bmatrix}\eta_1 & \dots & \eta_q\end{bmatrix}$, where $\eta_i = \mathbb{1}\{\theta_i > 0\}$, denote the support or sparsity pattern of $\boldsymbol{\Theta}$. That is, $\boldsymbol{\eta}(\boldsymbol{\Theta})$ represents the states that have at least one job to process. Note that there are only $2^q - 1$ of these different sparsity patterns.

Through simulations we noted that, just like in Theorem 6.1, the number of jobs in each θ_i does not determine the optimal policy, which only depends on the sparsity pattern of the state. In Theorem 6.1 we have already proved this result for $q = 2$ using double induction, but this method is not adequate to prove it for an arbitrary number of states q. Thus, we leave the observation as a conjecture.

Conjecture 6.1. *The optimal queue to process at state $\boldsymbol{\Theta}$ is the same when the state is $\boldsymbol{\eta}(\boldsymbol{\Theta})$.*

The importance of this conjecture is that, if true, it shows that given the job sizes $\boldsymbol{\Psi}$ and probability distribution $\mathbf{f}$, only computing the optimal rules for the $2^q - 1$ sparsity patters is needed to compute the optimal policy for any given state and thus for any given instance. Also note that since the size of each queue does not determine the policy, as long as there are jobs in the first queue, the offline and online optimal policies will be the same, since the arrival of a new job (which arrives to the first queue), will not make a difference and there will be no need for preemption.

Algorithm 6.1 SPARSITY RULE

Inputs: Set of jobs, $2^q - 1$ sparsity or support rules.

while $\mid \boldsymbol{\Theta} \mid > 0$
    process a job from queue θ_i, where i is the optimal queue for state $\boldsymbol{\eta}(\boldsymbol{\Theta})$.
    set $\theta_i = \theta_i - 1$
    if the job does not finish
        $\theta_{i+1} = \theta_{i+1} + 1$

Using this conjecture we propose the algorithm Sparsity Rule, detailed in Algorithm 6.1 to process an instance. The optimal policy can be computed for each of the $2^q - 1$ sparsity patterns in $O(q^q)$ computations by solving the DP with q jobs, but it is done only once. Hence, the optimal scheduling for any instance can be then computed in $O(n)$.

6.3.1.3 Experimental Results

We analyzes the performance of the Sparsity Rule algorithm using simulations with randomly generated parameters. For each simulation instance, we randomly selected $\mathbf{\Psi}$ and $\mathbf{f}$ from uniform distributions with up to $n = 15$ jobs, which was the limit of what the computer could handle. We did $20,000$ simulations for each value of q from 2 to 8 and in each simulation we compared the Sparsity Rule algorithm rule with the actual optimal policy from the DP solution. The simulations results showed that for all the $140,000$ simulations the rule given by the Sparsity Rule algorithm was exactly the same as th optima policy given by the DP formulation.

We further analyzed the setting when $q = 3$ to identify patters that might help us deduce simpler rules, like the ones achieved for th $q = 2$ case, for larger state spaces. To do this we defined a probability distribution of the job sizes and then randomly generated the size parameters. For each set of job sizes we computed the optimal rules for all the 4 different sparsity patterns. Figure 6.2 shows one specific set of simulations where $\mathbf{f} = \begin{bmatrix} 0.3 & 0.5 & 0.2 \end{bmatrix}$. The areas in green denote the instances where choosing a job from θ_3 was optimal, the red ones from θ_2, and the blue ones from θ_1.

Figure 6.2 shows that there is indeed some structure in the rules for each sparsity pattern, but it is not clear what is the correct hierarchy required to order them and extract the rules without having to solve a DP for q jobs.

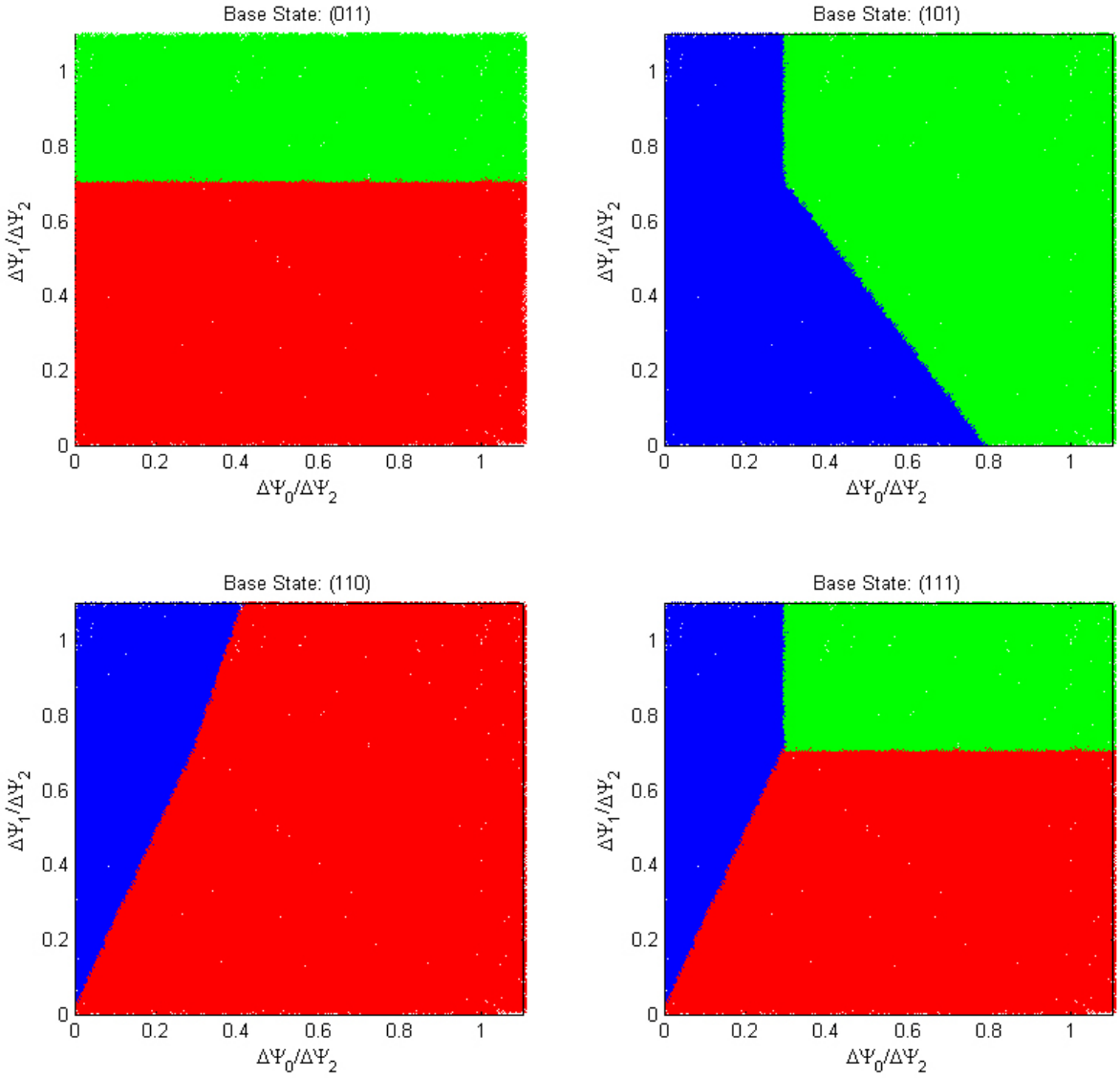


Figure 6.2: Rule Decisions for $q = 3$ and $\mathbf{f} = [0.3\ 0.5\ 0.2]$.

This page is intentionally printed only with this statement.

Chapter 7

Scheduling with Convex Costs

The key modification in the tardiness extension of the SAIAS algorithm presented in Section 3.2 is that we speed up jobs by a factor γ to make sure that early jobs in the optimal solution remain early in our approximation. This technique of increasing a resource (in this case speed) to achieve an approximate optimal solution is known as resource augmentation, and it has been recently used in many scheduling applications as a method to circumvent the limitations of the worst-case scenario analysis that is done in approximation algorithms. A summary of the results presented in this chapter can be found in [Carrasco *et al.*, 2013].

In resource augmentation algorithms, the approximation algorithm is allowed to use more resources (in this case a faster machine) than the optimal algorithm without paying for the extra resources used. Thus, it is equivalent to the problem described in Chapter 3 but with $\mathcal{E}_i(s_i) \equiv 0$, for all i. For this specific case we can extend its applicability to a very general class of scheduling problems, where the scheduling performance metric used is a convex non-decreasing function of the completion time of each job. This class includes important metrics such as weighted tardiness and completion time squared, which are widely used in various applications.

Although this application can be seen as an extension and corollary of the tardiness approximation algorithm in Section 3.2 because of its applicability and importance deserves

special attention and we will discuss it in further detain in this Chapter.

In this chapter, after reviewing the state-of-the-art in Section 7.1, we present the problem formulation in Section 7.2. After describing our algorithm, in Section 7.3 we analyse its approximation ratio. In doing so we combine techniques from the α-points literature as well as the resource augmentation literature. The key insight is that through resource augmentation, specifically speed scaling, we ensure that the completion times given by the algorithm and the optimal completion times are comparable.

Finally, in section 7.4 we report the results of our numerical experiments. These results clearly show that our algorithm performs very well and that the actual speed-scaling required is less than 2% on average.

7.1 Introduction

We consider the following offline scheduling problem: we are given a collection of n jobs, where job i has a requirement of ρ_i machine cycles, and the cost of completing job i at time C_i is given by a non-decreasing convex function $f_i : \mathbb{R}^+ \to \mathbb{R}^+$ of the completion time C_i. Additionally, we are given arbitrary precedence constraints and the objective is to compute a schedule that minimizes the total cost $\sum_i f_i(C_i)$. Since the jobs do not have release dates, we can assume, without loss of generality, that the schedule is non-preemptive. For simplicity we will use the three-field notation [Graham *et al.*, 1979], and denote this problem as $1|prec|\sum f_i(C_i)$.

Special cases of this problem have been studied since the early 1960s. Schild and Fredman considered $\sum_i C_i^2$ as the scheduling metric [Schild and Fredman, 1962], whereas later Townsend studied a more general cost functions of the form $f(C_i) = w_i C_i^2 + v_i C_i$, where w_i and v_i are arbitrary weights [Townsend, 1978]. More recently Höhn and Jacobs [Höhn and Jacobs, 2012] presented a 1.75-approximation algorithm for the $\sum_i w_i C_i^2$ problem. Some additional examples can be found in [Alidaee, 1991; Alidaee, 1993; Bagga and Kalra, 1980; Croce *et al.*, 1995; Gupta and Sen, 1984; Mondal and Sen, 2000; Moore, 1968]. The first

major breakthrough with general, possible non-convex, cost functions was due to Bansal and Pruhs [Bansal and Pruhs, 2010], who developed a $O(\log\log nP)$-approximation algorithm for the $1|r_i, pmtn|\sum f_i(C_i)$ problem. For the special case with no release dates (i.e. $r_i \equiv 0$) this algorithm is a 16-approximation algorithm. Later, Cheung and Shmoys [Cheung and Shmoys, 2011] presented a pseudo-polynomial $(2+\epsilon)$-approximation algorithm for the case without preemption. Note that, to the best of our knowledge, there are no results for problems with precedence constraints.

Due date related metrics are another set of interesting and widely used scheduling metrics that satisfy the convexity requirement [Pinedo, 2008]. In these problems each job i has an associated deadline d_i and the objective function of the problem is a function of the completion times C_i and the deadline d_i. Some typical examples of convex due date related scheduling metrics are lateness (defined as $L_i = C_i - d_i$) and tardiness (defined as $T_i = \max\{0, C_i - d_i\}$). Although this problem was first proposed in the early 1960s [McNaughton, 1959; Schild and Fredman, 1961] it has remained a challenging topic, since the total weighted tardiness problem is NP-Hard even in the case without release dates or precedence constraints [Lawler, 1977; Lawler and Moore, 1969].

The weighted tardiness scheduling metric is widely used in many industrial applications, including e.g. production plants, repair procedures, and routing schedules, etc., where deadlines are involved; and both scheduling metrics have received much attention in the scheduling literature [Pinedo, 2008]. Dynamic programming and branch and bound algorithms are the two main approaches that have been proposed for the total tardiness and total weighted tardiness problems. In both of these approaches, one adds dominance rules to reduce the state space and, consequently, speed up the algorithms [Abdul-Razaq *et al.*, 1990; Sen *et al.*, 2003]. Still, due to the size of the state space, these methods are only able to solve problems with approximately 50 jobs. Other search heuristics such as simulated annealing, genetic algorithms, etc. have also been proposed [Crauwels *et al.*, 1998; Potts and van Wassenhove, 1991]. To the best of our knowledge, only Bansal, et al. [Bansal *et al.*, 2007a] have addressed the total weighted tardiness from a resource augmentation point of

view, and no α-point based approximation algorithms have been proposed for this problem. Bansal et al. [Bansal *et al.*, 2007a] propose a 2-machine, 24-speed, 4-approximation algorithm for the $1|r_i|\sum w_i T_i$ problem.

The tardiness squared T_i^2 metric severely penalizes large tardiness values, making it a natural metric for processes where a just-in-time type of approach is needed. Only branch-and-bound algorithms have been proposed to solve the weighted tardiness squared problem [Schaller and Valente, 2012]. Furthermore, these approaches can only solve very small instances.

In this paper we propose an approximation algorithm for the scheduling problem with job-dependent non-decreasing convex cost functions and arbitrary precedence constraints that builds on techniques from the α-point and resource augmentation literatures.

The α-point algorithms were introduced by Phillips, Stein, and Wein [Phillips *et al.*, 1998], and Hall, Schulz, Shmoys, and Wein [Hall *et al.*, 1997; Hall *et al.*, 1996], and have resulted in small constant factor approximation algorithms for many scheduling problems [Skutella, 2006]. In this approach, the scheduling problem is formulated as a binary integer program (IP) in terms of decision variables x_{it} that is set to 1 if job i completes at time t and 0 otherwise. The α-point of each job is defined as the earliest time at which an α fraction of the job has been completed in linear relaxation of the IP. The jobs are then ordered in some fashion according to these α-points. Currently there are many variants and extensions of these technique including choosing α randomly [Chekuri *et al.*, 2001; Goemans, 1997] or choosing a different α for each job [Goemans *et al.*, 2002]. A detailed survey of many of the current α-point algorithms and the approximation ratios achieved by them can be found in [Skutella, 2006].

Resource augmentation was developed to circumvent the shortcomings of the worst-case approximation ratio criterion, and has helped explain why some algorithms perform much better in practice than their worst-case theoretical guarantees. The main idea of this analysis is that one compares the optimal solution of the original problem with the solution computed by an algorithm that has access to an augmented set of resources: more machines, more space,

faster speed, etc. Although the first examples of resource augmentation analysis are almost three decades old [Sleator and Tarjan, 1985], this technique has become popular only recently. Kalyanasundaram and Pruhs [Kalyanasundaram and Pruhs, 2000] introduced the idea, by showing that running the algorithm at a faster speed was equivalent to having clairvoyance in the total flow time scheduling problem. Although it is known that there are no constant factor non-clairvoyant on-line algorithm for this scheduling problem, they showed that one can achieve a constant competitive ratio by speed scaling. The term *resource augmentation analysis* was introduced by Philips, Stein, Trong, and Wein [Phillips *et al.*, 2002]. They defined an s-speed ρ-approximation algorithm, as an algorithm that achieves a ρ worst-case approximation ratio when jobs run at s times the nominal speed of the machine. A survey including on-line resource augmentation results can be found in [Pruhs *et al.*, 2004].

7.1.1 Our Results

We make several contributions to the problem of scheduling jobs with non-decreasing convex cost functions:

(a) We introduce a model that extends the previous models by allowing a more general non-linear job-dependent function of the completion time as the scheduling metric.

(b) We propose a new approximation algorithm for minimizing the total cost, with arbitrary precedence constraints.

(c) Our algorithm builds on both the α-point and resource augmentation techniques. We show that our algorithm has a small constant approximation ratio and a small speed-scaling ratio for several important scheduling metrics, namely the total weighted tardiness, the total weighted tardiness squared, and the total completion time squared. The results of our numerical experiments show that the practical performance of our algorithm is significantly superior to the theoretical bounds.

(d) We compare the performance of our algorithm with other available methods for the total weighted tardiness problem by using the test instances from the OR Library [Beasley,

1990]. We show that our algorithm is capable of computing approximate optimal solutions for all the available test problems, even those with $n = 100$ jobs. Thus, we are able to establish lower bounds on the optimal solutions for instances where the optimal schedule is currently not known. Our algorithm takes less than a second to solve even the larger instances. This is at least one order of magnitude faster than current methods. Furthermore, we show that on average only a 2% speed-up is required to achieve the best known result; and, in fact, in several cases no speed-up factor is required.

Our main result can be summarized in the following theorem

Theorem 7.1. *Given n jobs with arbitrary precedence constraints and convex non-decreasing cost functions $f_i(C_i)$ for each job i, there is a $O(1)$-speed 1-approximation algorithm for the problem of minimizing the total non-linear cost $\sum f_i(C_i)$.*

The speed scaling constant is relatively small. Given $\epsilon > 0$ our algorithm is a $(4+\epsilon)$-speed 1-approximation algorithm.

7.1.2 Our Methodology

Time-indexed IP formulations for scheduling problems are not typically of polynomial size, therefore, we extend the polynomial size *interval-indexed* IP formulation introduced in [Hall *et al.*, 1997], to handle the non-linear cost functions. The basic idea is to divide time into geometrically increasing intervals and assign the completion time of each job to one of these intervals instead of a specific time. In this formulation, variables x_{it} are 1 if job i is completed in interval t and 0 otherwise. This allows a polynomially sized problem, but since completion times only belong to an interval and not exactly known, the approximation ratio suffers a small degradation. In Section 7.2 we describe this interval-indexed formulation in further detail.

7.2 Problem Formulation

7.2.1 Problem Setting

The problem setting is as follows. We are given n jobs, where job i has a processing requirement of $\rho_i \in \mathbb{N}_+$ machine cycles, and thus requires $p_i = \frac{\rho_i}{\sigma}$ time units on a machine that runs at speed of σ cycles per time unit. Without loss of generality, we assume that $\rho_i > 0$, for all $i = \{1, \ldots, n\}$ and $\sigma = 1$. Let C_i denote the completion time of job i and $f_i(C_i)$, with $f_i : \mathbb{R}_+ \to \mathbb{R}_+$, denote a job-dependent non-decreasing convex cost function. Let $\Pi = \{\pi(1), \ldots, \pi(n)\}$ denote the order in which the jobs are processed, i.e. $\pi(i) = k$ implies that the i-th job to be processed is job k. Then $C_{\pi(i)} = C_{\pi(i-1)} + p_{\pi(i)}$ is the completion time of the i-th job, where $\pi(0) = 0$ and $C_{\pi(0)} = 0$, and is completely determined by the order Π when all jobs are processed at the same speed.

The objective is to compute a feasible schedule, consisting of an order Π that respects precedence constraints and minimizes the total cost of all jobs, i.e. minimizes the function, $F(\Pi) = \sum_{i=1}^{n} f_{\pi(i)}\left(C_{\pi(i)}\right)$.

7.2.2 Interval-Indexed Formulation

We now modify and extend the interval-indexed formulation proposed in Chapter 3 for the total non-linear cost setting. The interval-indexed formulation divides the time horizon into geometrically increasing intervals, and the completion time of each job is assigned to one of these intervals. The problem formulation is as follows. We divide the time horizon into the following geometrically increasing intervals: $[\kappa, \kappa]$, $(\kappa, (1+\epsilon)\kappa]$, $((1+\epsilon)\kappa, (1+\epsilon)^2\kappa]$, ..., where $\epsilon > 0$ is an arbitrary small constant, and $\kappa = \min_j\{p_j\}$ denotes the smallest interval size that will hold at least one whole job. We define interval $I_t = (\tau_{t-1}, \tau_t]$, with $\tau_0 = \kappa$ and $\tau_t = \kappa(1+\epsilon)^{t-1}$. The interval index ranges over $\{1, \ldots, T\}$, with $T = \min\{\lceil t \rceil : \kappa(1+\epsilon)^{t-1} \geq \sum_{i=1}^{n} p_i\}$; and thus, we have a polynomial number of indices t.

Let x_{it} be 1 if job i completes in the time interval $I_t = (\tau_{t-1}, \tau_t]$ and 0 otherwise. We consider the following IP, which is a lower bound on $F(\Pi)$ since we consider the initial time

of each time interval I_t:

$$\min_{\mathbf{x}} \quad \sum_{i=1}^{n} \sum_{t=1}^{T} f_i(\tau_{t-1})\, x_{it} \tag{7.1}$$

$$\text{s.t.:} \quad \sum_{t=1}^{T} x_{it} = 1, \qquad i = \{1, \ldots, n\}, \tag{7.2}$$

$$\sum_{i=1}^{n} \sum_{u=1}^{t} p_i x_{iu} \leq \tau_t, \qquad t = \{1, \ldots, T\}, \tag{7.3}$$

$$x_{it} = 0, \text{ if } \tau_t < p_i, \qquad i = \{1, \ldots, n\}, t = \{1, \ldots, T\}, \tag{7.4}$$

$$\sum_{u=1}^{t} x_{i_1 u} \geq \sum_{u=1}^{t} x_{i_2 u}, \text{ if } i_1 \prec i_2, \quad t = \{1, \ldots, T\}, \tag{7.5}$$

$$x_{it} \in \{0, 1\}, \qquad i = \{1, \ldots, n\}, t = \{1, \ldots, T\}. \tag{7.6}$$

We have constraint (7.2) because each job i must finish in a unique time interval; constraint (7.3) because since only one job can be processed at any given time, the total processing time of jobs up to time interval I_t must be at most τ_t units; constraint (7.4) because each job i requires p_i time units to be processed; and constraint (7.5) because the precedence constraint $i_1 \prec i_2$ implies that job i_2 cannot finish in an interval earlier than i_1. Note that, although the function $f_i(C_i)$ is non-linear, the objective in the interval approximation (7.1) is linear and also note that the error associated with the interval relaxation is controlled through ϵ.

It is important to note that this integer program only provides a lower bound for $F(\Pi)$ but it might not be feasible. This is because the precedence constraints (7.5) do not ensure that two sequential jobs that finish in the same interval have the right order.

7.3 Approximation Algorithm

We now describe the speed scaling and α-point based algorithm for the total job-dependent non-linear cost metric, called Schedule by α-intervals and Resource Augmentation (SAIRA) which is detailed in Algorithm 7.1.

Let $\bar{x}_{it}$ denote the optimal solution of the linear relaxation of the integer program (7.1) when replacing the constraints (7.6) by $x_{it} \geq 0$. In steps 1 and 2 of the algorithm we model and compute the optimal solution $\bar{\mathbf{x}}$ and in step 3, given $0 \leq \alpha \leq 1$, we compute the α-interval of job i, which is defined as, $I_i^{\alpha} = \min\left\{t : \sum_{u=1}^{t} \bar{x}_{iu} \geq \alpha\right\}$. Since several jobs may finish in

Algorithm 7.1 SCHEDULE BY α-INTERVALS AND RESOURCE AUGMENTATION (SAIRA)

Inputs: set of jobs, functions $f_i(C_i)$, $\alpha \in (0,1)$, $\epsilon > 0$, $\gamma > 1$.

1 Divide time into increasing time intervals $I_t = (\tau_{t-1}, \tau_t]$, with $\tau_t = \kappa(1+\epsilon)^{t-1}$.

2 Compute an optimal solution $\bar{\mathbf{x}}$ to the linear relaxation of problem (7.1).

3 Compute the α-intervals $\mathbf{I}^\alpha$ and the sets J_t.

4 Compute an order Π^α that has the sets J_t ordered in non-decreasing values of t and the jobs within each set in a manner consistent with the precedence constraints.

5 Run each job i at speed γ.

6 Set the i-th job to start at time $C^\alpha_{\pi(i-1)}$, which is the completion time of the previous job using speeds γ, and $C^\alpha_{\pi(0)} = 0$.

7 **return** schedule Π^α and completion times $\mathbf{C}^\alpha$.

the same interval, let $J_t = \{i : I^\alpha_i = t\}$ denote the set of jobs that finish in interval I_t. In step 4 we describe the order Π^α that we use to schedule jobs.

Next, in step 5, we speed up each job by γ and finally in steps 6 and 7 we compute the completion times given the calculated speeds and return the schedule Π^α and completion times $\mathbf{C}^\alpha$. Note that since γ is constant for all jobs, the SAIRA algorithm is a γ-speed approximation algorithm.

We now analyse this algorithm's performance. We will assume, without loss of generality, that $I^\alpha_1 \leq I^\alpha_2 \leq \ldots I^\alpha_n$.

First, the following lemma shows that the output of the SAIRA algorithm is indeed feasible.

Lemma 7.1. *If $i_1 \prec i_2$, then constraint (7.5) implies that $I^\alpha_{i_1} \leq I^\alpha_{i_2}$.*

Proof. Evaluating constraint (7.5) corresponding to $i_1 \prec i_2$, for $t = I^\alpha_{i_2}$, we have that,

$$\sum_{u=1}^{I^\alpha_{i_2}} x_{i_1 u} \geq \sum_{u=1}^{I^\alpha_{i_2}} x_{i_2 u} \geq \alpha,$$

where the last inequality follows from the definition of $I^\alpha_{i_2}$. Since the chain of inequalities implies that $\sum_{u=1}^{I^\alpha_{i_2}} x_{i_1 u} \geq \alpha$, which means that the α-interval for i_1 must satisfy that $I^\alpha_{i_1} \leq I^\alpha_{i_2}$. □

Since the SAIRA algorithm schedules jobs by first ordering the sets J_t in increasing order of t and then orders the jobs within each set in a way that is consistent with the precedence constraints, by Lemma 7.1 it follows that the SAIRA algorithm preserves the precedence constraints, and, therefore, the output of the algorithm is feasible.

Theorem 7.2. *The SAIRA algorithm with $\alpha = \frac{1}{2}$ and $\gamma = (4+\epsilon)$ is a $(4+\epsilon)$-speed, 1-approximation algorithm for the total weighted non-linear cost problem with convex non-decreasing cost and arbitrary precedence constraints, $1|prec|\sum f_i(C_i)$.*

Proof. Let C_i^* be the completion time of job i in the optimal solution, $f_i(C_i^*)$ denote its cost, $\hat{x}_{it}$ denote an optimal solution of the integer problem (7.1), and $\bar{x}_{it}$ the fractional optimal solution of its linear relaxation. Because we consider τ_{t-1} as the completion time for all jobs that finish in time interval t, and since $\bar{x}_{it}$ is the optimal solution of the linear relaxation and $f_i(C)$ is non-decreasing we have that

$$\sum_{i=1}^{n}\sum_{t=1}^{T} f_i(\tau_{t-1})\,\bar{x}_{it} \le \sum_{i=1}^{n}\sum_{t=1}^{T} f_i(\tau_{t-1})\,\hat{x}_{it} \le \sum_{i=1}^{n} f_i(C_i^*). \tag{7.7}$$

Let $\bar{C}_i = \sum_{t=1}^{T} \tau_{t-1}\bar{x}_{it}$ denote the optimal fractional completion time of job i, given by the optimal solution of the relaxed linear program. Because $f_i(C)$ is convex and $\sum_t \bar{x}_{it} = 1$ from constraint (7.2) then

$$f_i\left(\bar{C}_i\right) = f_i\left(\sum_{t=1}^{T}\tau_{t-1}\bar{x}_{it}\right) \le \sum_{t=1}^{T} f_i(\tau_{t-1})\,\bar{x}_{it}. \tag{7.8}$$

and thus, from (7.7) and (7.8),

$$\sum_{i=1}^{n} f_i\left(\bar{C}_i\right) \le \sum_{i=1}^{n} f_i(C_i^*). \tag{7.9}$$

Because there are no release date constraints there is no idle time between jobs and we can bound the completion time determined by the algorithm, C_i^α, by

$$C_i^\alpha = \frac{1}{\gamma}\sum_{j=1}^{i} p_j \le \frac{1}{\gamma\alpha}\sum_{j=1}^{i}\sum_{u=1}^{I_j^\alpha} p_j\bar{x}_{ju} \le \frac{1}{\gamma\alpha}\sum_{j=1}^{n}\sum_{u=1}^{I_i^\alpha} p_j\bar{x}_{ju}, \tag{7.10}$$

and from constraint (7.3) for $t = I_i^\alpha$ we get that,

$$C_i^\alpha \leq \frac{1}{\gamma\alpha}\tau_{I_i^\alpha}. \tag{7.11}$$

Since it is possible that $\sum_{t=1}^{I_i^\alpha} \bar{x}_{it} > \alpha$; we define $X_i^{(1)} = \alpha - \sum_{t=1}^{I_i^\alpha - 1} \bar{x}_{it}$ and $X_i^{(2)} = \sum_{t=1}^{I_i^\alpha} \bar{x}_{it} - \alpha$, thus $X_i^{(1)} + X_i^{(2)} = \bar{x}_{iI_i^\alpha}$, and we can rewrite

$$\bar{C}_i = \sum_{t=1}^{I_i^\alpha - 1} \tau_{t-1}\bar{x}_{it} + \tau_{I_i^\alpha - 1}X_i^{(1)} + \tau_{I_i^\alpha - 1}X_i^{(2)} + \sum_{t=I_i^\alpha+1}^{T} \tau_{t-1}\bar{x}_{it}, \tag{7.12}$$

and eliminating the lower terms of the previous sum we get that,

$$\bar{C}_i \geq \tau_{I_i^\alpha - 1}X_i^{(2)} + \sum_{t=I_i^\alpha+1}^{T} \tau_{t-1}\bar{x}_{it} \geq \tau_{I_i^\alpha - 1}X_i^{(2)} + \sum_{t=I_i^\alpha+1}^{T} \tau_{I_i^\alpha - 1}\bar{x}_{it} = \tau_{I_i^\alpha - 1}(1-\alpha). \tag{7.13}$$

Because $\tau_{I_i^\alpha} = (1+\epsilon)\tau_{I_i^\alpha - 1}$, from (7.11) and (7.13) we get that $C_i^\alpha \leq \frac{(1+\epsilon)}{\gamma\alpha(1-\alpha)}\bar{C}_i$. The key step is that by setting $\gamma = \frac{(1+\epsilon)}{\alpha(1-\alpha)}$, which implies that we have a $\frac{(1+\epsilon)}{\alpha(1-\alpha)}$-speed approximation algorithm, we make the algorithm's completion time and the fractional completion time comparable, and thus,

$$\sum_{i=1}^{n} f_i\left(C_i^\alpha\right) \leq \sum_{i=1}^{n} f_i\left(\frac{(1+\epsilon)}{\gamma\alpha(1-\alpha)}\bar{C}_i\right) = \sum_{i=1}^{n} f_i\left(\bar{C}_i\right). \tag{7.14}$$

Finally, from (7.9) and (7.14) it follows that

$$\sum_{i=1}^{n} f_i\left(C_i^\alpha\right) \leq \sum_{i=1}^{n} f_i\left(C_i^*\right), \tag{7.15}$$

and by setting $\alpha = \arg\min_{0\leq\alpha\leq 1}\left\{\frac{1}{\alpha(1-\alpha)}\right\} = \frac{1}{2}$, which minimizes the resource augmentation requirement, we get the desired speed and approximation ratios. □

Regretfully, we are not able to extend this algorithm to the setting where release dates are present, since we use speed scaling as a method to make sure that jobs are finished in a certain interval.

Since the tardiness metric to any power $d \geq 1$, that is $f_i\left(C_i\right) = T_i^d$, is convex and non-decreasing, we have the following corollary directly from the previous theorem.

Corollary 7.1. *For any $d \geq 1$, the SAIRA algorithm with $\alpha = \frac{1}{2}$ and $\gamma = (4+\epsilon)$ is a $(4+\epsilon)$-speed, 1-approximation algorithm for the total weighted tardiness with arbitrary precedence constraints, $1|prec|\sum w_i T_i^d$.*

For the specific setting where $f_i(C_i) = w_i C_i^d$, with $d \geq 1$, if no resource augmentation is allowed, we have the following additional corollary.

Corollary 7.2. *For any $d \geq 1$, the SAIRA algorithm with $\alpha = \frac{1}{2}$ and $\gamma = 1$ is a $(4+\epsilon)^d$-approximation algorithm for the $1|prec|\sum w_i C_i^d$ problem.*

Proof. Without resource augmentation, from Theorem 7.2, we still have that

$$\sum_{i=1}^{n} f_i\left(C_i^{\alpha}\right) \leq \sum_{i=1}^{n} f_i\left(\frac{(1+\epsilon)}{\gamma\alpha(1-\alpha)}\bar{C}_i\right), \tag{7.16}$$

where $f_i(C_i) = C_i^d$. Thus,

$$\sum_{i=1}^{n} \left(C_i^{\alpha}\right)^d \leq \sum_{i=1}^{n} \left(\frac{(1+\epsilon)}{\gamma\alpha(1-\alpha)}\bar{C}_i\right)^d = \sum_{i=1}^{n} \left(\frac{(1+\epsilon)}{\gamma\alpha(1-\alpha)}\right)^d \left(\bar{C}_i\right)^d, \tag{7.17}$$

and from (7.9) it follows that

$$\sum_{i=1}^{n} \left(C_i^{\alpha}\right)^d \leq \sum_{i=1}^{n} \left(\frac{(1+\epsilon)}{\gamma\alpha(1-\alpha)}\right)^d \left(C_i^*\right)^d. \tag{7.18}$$

By taking $\gamma = 1$, which implies that no resource augmentation is required, and $\alpha = \frac{1}{2}$, we obtain the corresponding approximation bound. □

7.4 Experimental Results

In this section we present a simulation based performance analysis of the SAIRA algorithm. In our simulations we considered three common scheduling metrics: total weighted tardiness, where $f_i(C_i) = w_i(C_i - d_i)^+$, with d_i denoting job i's due date; total weighted tardiness squared, where $f_i(C_i) = w_i\left((C_i - d_i)^+\right)^2$; and total weighted completion time squared, where $f_i(C_i) = w_i C_i^2$.

For each scheduling metric we simulated a large number of randomly generated instances with $\epsilon = 0.1$ and following the guidelines in [Hall and Posner, 2001] for deadlines values and precedence constraints. We used the following distributions: $w_i \sim \text{unif}\{0, \ldots, 20\}$, $\rho_i \sim \text{unif}\{1, \ldots, 10\}$, and for the tardiness metrics $d_i \sim \text{unif}\{0, \ldots, 0.1 \sum \rho_i\}$. We also analysed instances with much larger job sizes (such as $\rho_i \sim \text{unif}\{1, \ldots, 100\}$), as well as ρ_i drawn from bimodal distributions, which are generally hard for scheduling algorithms, without observing any significant degradation in the performance of our algorithm.

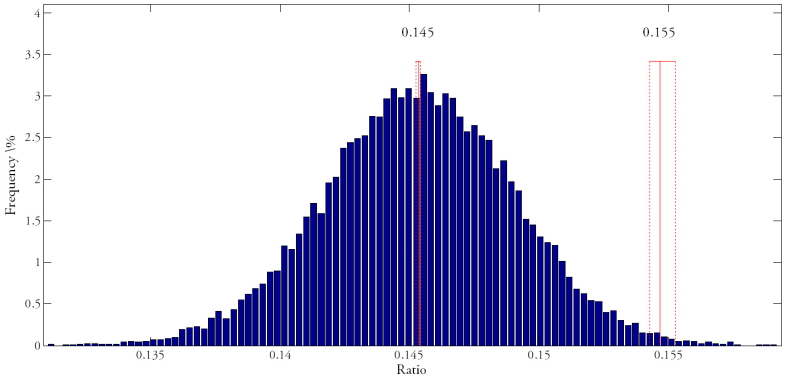
We compared the output of our algorithm with the integer solution of the interval-and-speed-indexed formulation (IPi), its linear relaxation (LPi), and when possible to the integer and relaxed solutions of a time-indexed formulation for this problem (IPt and LPt respectively). Although we do not explicitly provide these two last formulations in this paper, we use them to help us understand whether the optimality gap is a consequence of the rounding in the algorithm or due to the gap in the interval relaxation of the LP. Since the IPt formulation is non-polynomial in size, it is impractical for large instances, and thus, we have to use the LPi solution as a proxy to estimate our empirical approximation ratio. Hence, the real approximation ratio is likely to be better. All simulations were done in Matlab, using

Problem	Instances	Size (n)	Average Ratio	99.5%	Worst Ratio
$1\|prec\|\sum w_i T_i$	$20,720$	10	0.139	0.189	0.205
	$20,000$	10, bimodal	0.150	0.220	0.222
	$20,010$	50	0.151*	0.179	0.192
	$20,010$	100	0.148*	0.168	0.176
	$20,002$	500	0.145*	0.155	0.159
$1\|prec\|\sum w_i C_i^2$	$20,010$	10	0.052	0.053	0.055
	$20,010$	100	0.059*	0.059	0.059
$1\|prec\|\sum w_i T_i^2$	$20,000$	10	0.023	0.036	0.041
	$20,000$	500	0.028*	0.031	0.032

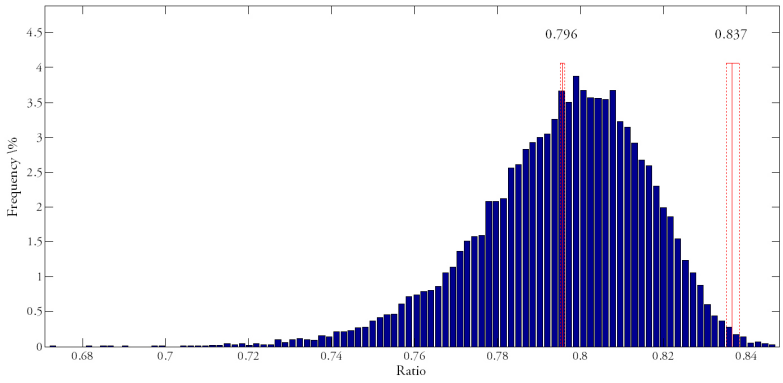
Table 7.1: Experimental Approximation Ratios Summary

Gurobi [Gurobi Optimization Inc., 2012] and Gurobi MEX [Yin, 2012] to solve the IP and LP relaxations of each instance. Table 7.1 shows a summary of simulation results for all the different cases and instance sizes. For large instances ($n \geq 50$) the ratios were computed comparing the algorithm's output with the LPi and not the IPt solution*.

Figure 7.1a shows further details for the total weighted tardiness case with instances of $n = 500$ jobs. In this figure we display the full histogram of the approximation ratios. We believe that the full histogram gives a better understanding of how the algorithm performs as compared to just reporting an average value or a worst case value. In the histogram we highlight the average value and the 99.5% quantile, and, we also display the 99.99% confidence intervals using dotted lines.



(a) SAIRA/LPi Ratio for $n = 500$, $20,002$ instances.



(b) LPi/IPt Ratio for $n = 10$, $20,720$ instances.

Figure 7.1: SAIRA Experimental Approximation Ratio for $1|prec|\sum w_i T_i$.

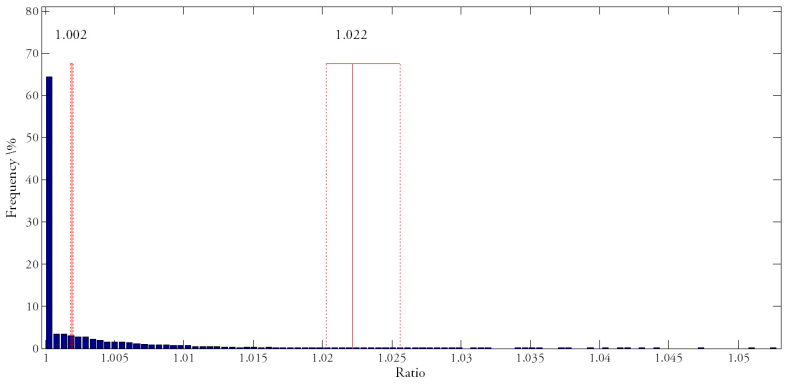


Figure 7.2: Minimum Speed-Up Required for $1|prec|\sum w_i T_i$, for $n = 10$.

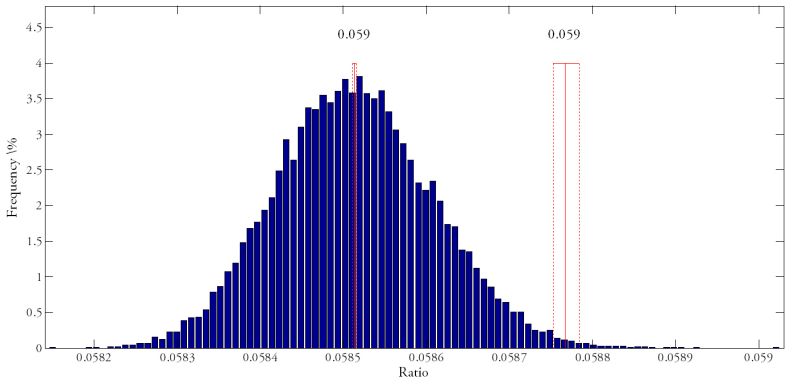
Although the performance shown in Figure 7.1a is very good, the real approximation ratio is likely to be even better since, as shown in Figure 7.1b, the average gap between the IPt and LPI solutions is 0.8. Additionally, we computed the minimum speed-up required in each instance so that the schedule determined by the SAIRA algorithm would have the same cost as the optimal schedule. The results displayed in Figure 7.2 shows that only a small speed-up is required – the maximum speed-up required over all the instances is only 5.3%.

We repeated the same analysis for the $1|prec|\sum w_i C_i^2$ problem. Figures 7.3a and 7.3b show details of some of the simulated settings. Since the cost function is quadratic in C_i, the performance is even better than in the total weighted tardiness case and the required speed-up is smaller, with a maximum speed-up of 1.035. As expected, when the T_i^2 metric was used the performance was better than when using the T_i metric, as shown in Table 7.1. Since in both cases the cost functions are quadratic, less speed-up is required to reduce the cost of the schedule when compared to the linear cost functions.

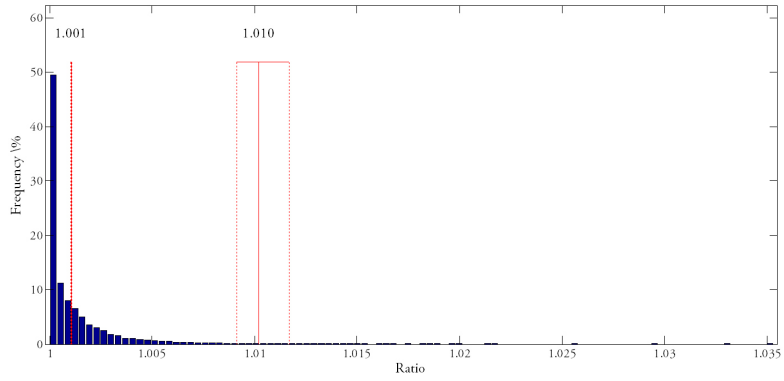
Next, we report the performance of our algorithm on the weighted tardiness test problems [Crauwels *et al.*, 1998] in the OR Library [Beasley, 1990], that are widely used in the tardiness literature to compare algorithm performance and optimal values. The problems consist of 3 sets with $n = 40$, 50, and 100 jobs per instance, with 125 instances each. The problem set has the optimal solutions for 124 of the $n = 40$ problems, and 115 of the $n = 50$ ones. For the remaining unsolved instances, the problem set has the best known schedules. Note

that these test problems do not have any precedence constraints. Just as in our randomly generated instances the problems where solved in Matlab, using Gurobi 4.6 as LP solver in a 4-core second generation i3 Intel CPU 550, with 6 Gb in RAM running Windows 7. For each instance we compared the algorithm's solution value with the best known solution for that problem in order to compute the empirical approximation ratio. We also computed the minimum speed-up required to achieve the best known solution and the CPU time required to achieve to that solution. Table 7.2 shows average and worst results for each problem set.

As expected from the previous experiments the approximation ratios are very small, but more importantly the minimum required speed-up is below 2% on average and below 8.6% in the worst case, i.e. speeding-up the machine by a factor of 2%, on average, ensures that the



(a) SAIRA/LPi Ratio for $n = 100,\ 20,010$ instances.



(b) Minimum Speed-Up Required for $n = 10$.

Figure 7.3: SAIRA Experimental Approximation Ratio for $1|prec|\sum w_i C_i^2$

Size (n)	Approx. Ratio		Min. Speed-up		CPU Time (sec.)	
	avr.	worst	avr.	worst	avr.	worst
40	0.077	0.22	1.018	1.079	0.150	0.270
50	0.075	0.22	1.020	1.086	0.202	0.447
100	0.080	0.21	1.011	1.048	0.486	0.885

Table 7.2: Experimental Approximation Ratios Summary

schedule computed by our algorithm has the same value as the best known schedule for that instance. For 25 instances (7 of which belong to the $n = 100$ set) the algorithm computed a schedule that required no speed-up to achieve the current best known solution.

Another important thing to highlight is that our algorithm is able to compute approximate solutions in under 1 second for all instances; even those with $n = 100$, some of which cannot be solved using state-of-the-art brand-and-bound and dynamic programming algorithms. Even for the smaller instances ($n = 40$), our algorithm requires about two orders of magnitude less time to compute approximate solutions; a gap that becomes even larger as the size of the instance increases [Schaller and Valente, 2012].

This page is intentionally printed only with this statement.

Chapter 8

Conclusions

THROUGHOUT this work we have introduced a general model for scheduling jobs with job-dependent non-renewable resources and we have given a small constant-factor approximation algorithm for minimizing the sum of weighted completion time and the total resource cost. We built this algorithm from a simpler setting in which only energy is the available resource which determines at which speed each job runs.

We further analysed the setting of energy aware scheduling, identifying cases that can be solved in polynomial time, as well as describing heuristic improvements to our algorithm. We have also tested the performance of the algorithm through simulations, showing that the SAIAS algorithm's output is very close to optimum. Furthermore, we tested the algorithm in additional settings, such as on-line, using the total weighted flow time as scheduling metric, and considering multiple machines, showing that the performance is very good in all these settings as well.

Additionally, we studied the case where job sizes are unknown but drawn from a known probability distribution. In this setting we present a linear time algorithm, in the number of jobs, that we proved to be optimal for the two queue setting. We also showed through simulations that it appears to be optimal for an arbitrary number of queues. Further analysis is required to prove that this is actually the case.

We have also described how to combine two known and successful techniques (α-points and

speed-scaling) to construct approximation algorithms for non-linear convex cost scheduling problems. To the best of our knowledge, our algorithm is the first $O(1)$-speed 1-approximation algorithm for the total weighted tardiness problem with arbitrary precedence constraints and we suspect that the speed-scaling requirement can be further improved. Furthermore, we showed through experimental analysis and using data from the OR Library, that our algorithm performs much better than the theoretical results suggest and that the actual speed-scaling required is very close to 1.

8.1 Future Research Directions

There are several open research directions that extend from the work presented here. For the resource cost aware scheduling problem, we believe that our methodology, which extends the idea of α-points to the resource cost aware setting by developing the α-speeds concept, should have many more applications. For example, by adding minor regularity conditions to the resource cost $\mathcal{R}(\mathbf{\Psi}^{(j)})$ and speed-scaling the resulting output, we can use the SAIAS algorithm in the setting where there are no release dates and the scheduling metric is a convex function of the completion time, like weighted tardiness or completion time squared.

We also suspect that, via techniques such as using randomly chosen values of α or using different α values for different jobs, we could obtain tighter bounds, and that these techniques could be extended to other settings, such as multiple parallel machines among others. Furthermore, since the result we obtain for the heuristic speed improvement is the same as the best known speed policy for the flow time setting, we believe that this result could help us determine the right speed policy for the weighted flow time setting which currently has no known approximation algorithms.

Additionally, we leave open the complexity of the energy aware setting in which all deadlines are the same. Although we observed though simulations that a simple ordering rule, similar to Smith's Rule, results in an optimal schedule every time, we need to prove that this is actually the case. There are no complexity results in this type of energy aware

scheduling problems, hence, a result like this could lead to determine which problems in the EAS literature are NP-Hard and which are not.

In the setting where jobs sizes are uncertain but come from a known distribution there are also several open questions. Although we proved that our algorithm is optimal when only two queues exist, the method used to prove it is not easily extended to multiple queues. We saw through simulations that this was always the case, so there might be some other methodology with which we can prove it and thus, show that our algorithm is optimal for the general case. Furthermore, we did not explore the use of this algorithm with other scheduling metrics, nor with more complex energy cost functions. We suspect that, with some modifications, our results could be applied to other scheduling metrics such as tardiness.

As for the convex cost scheduling problem, there are also several future research directions. The methodology presented in Chapter 7 could be further extended to other α-point based algorithms to obtain smaller speed scaling ratios for the scheduling problems presented here, or even construct new algorithms for other settings such as multiple machines or other scheduling metrics. Furthermore, we believe that the insights from the structure of the approximate solutions could be used to design dominance rules for the dynamic programming and/or branch and bound algorithms that are currently being used for total weighted tardiness and total weighted tardiness squared.

This page is intentionally printed only with this statement.

Bibliography

[Abdul-Razaq *et al.*, 1990] T.S. Abdul-Razaq, C.N. Potts, and Luk N. Van Wassenhove. A survey of algorithms for the single machine total weighted tardiness scheduling problem. *Discrete Applied Mathematics*, 26:235–253, 1990.

[Albers and Fujiwara, 2007] Susanne Albers and Hiroshi Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Transactions on Algorithms*, 3(4):49–es, November 2007.

[Albers, 2009] Susanne Albers. Algorithms for Energy Saving. In Susanne Albers, Helmut Alt, and Stefan Näher, editors, *Efficient Algorithms*, volume 5760 of *Lecture Notes in Computer Science*, pages 173–186. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[Albers, 2010] Susanne Albers. Energy-efficient algorithms. *Communications of the ACM*, 53(5):86, May 2010.

[Alidaee, 1991] Bahram Alidaee. Single machine scheduling with nonlinear cost functions. *Computers & operations research*, 18(3):317–322, 1991.

[Alidaee, 1993] Bahram Alidaee. Numerical Methods for Single Machine Scheduling with Non-Linear Cost Functions to Minimize Total Cost. *The Journal of the Operational Research Society*, 44(2):125, February 1993.

[Andrew *et al.*, 2009] Lachlan L.H. Andrew, Adam Wierman, and Ao Tang. Optimal speed scaling under arbitrary power functions. *ACM SIGMETRICS Performance Evaluation Review*, 37(2):39, October 2009.

[Andrew *et al.*, 2010] Lachlan L.H. Andrew, Minghong Lin, and Adam Wierman. Optimality, fairness, and robustness in speed scaling designs. *ACM Sigmetrics*, 2010.

[Atkins *et al.*, 2011] Leon Atkins, Guillaume Aupy, Daniel Cole, and Kirk R. Pruhs. Speed Scaling to Manage Temperature. *Lecture Notes in Computer Science*, 6595:9–20, 2011.

[Augustine *et al.*, 2004] J. Augustine, S. Irani, and C. Swamy. Optimal Power-Down Strategies. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 530–539. IEEE, 2004.

[Bagga and Kalra, 1980] P.C. Bagga and K.R. Kalra. NoteA Node Elimination Procedure for Townsend's Algorithm for Solving the Single Machine Quadratic Penalty Function Scheduling Problem. *Management Science*, 26(6):633–637, 1980.

[Bansal and Pruhs, 2005] Nikhil Bansal and Kirk R. Pruhs. Speed scaling to manage temperature. *Lecture Notes in Computer Science*, 3404:460–471, 2005.

[Bansal and Pruhs, 2010] Nikhil Bansal and Kirk Pruhs. The Geometry of Scheduling. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 407–414. IEEE, October 2010.

[Bansal *et al.*, 2004] Nikhil Bansal, Tracy Kimbrel, and Kirk R. Pruhs. Dynamic speed scaling to manage energy and temperature. *Energy*, 2004.

[Bansal *et al.*, 2007a] Nikhil Bansal, Ho-Leung Chan, Rohit Khandekar, Kirk Pruhs, Cliff Stein, and Baruch Schieber. Non-Preemptive Min-Sum Scheduling with Resource Augmentation. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 614–624. IEEE, October 2007.

[Bansal *et al.*, 2007b] Nikhil Bansal, Tracy Kimbrel, and Kirk R. Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM (JACM)*, 54(1):3, March 2007.

[Bansal *et al.*, 2008] Nikhil Bansal, D.P. Bunde, Ho Leung Chan, and Kirk R. Pruhs. Average rate speed scaling. In *Proceedings of the 8th Latin American conference on Theoretical informatics*, pages 240–251. Springer-Verlag, December 2008.

[Bansal *et al.*, 2009] Nikhil Bansal, Ho Leung Chan, and Kirk R. Pruhs. Speed scaling with an arbitrary power function. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 693–701. Society for Industrial and Applied Mathematics, 2009.

[Bansal *et al.*, 2010] Nikhil Bansal, Ho Leung Chan, T.W. Lam, and L.K. Lee. Scheduling for speed bounded processors. *Automata, Languages and Programming*, pages 409–420, 2010.

[Beasley, 1990] J. E. Beasley. OR-Library. Online at http://people.brunel.ac.uk/~mastjjb/jeb/orlib/wtinfo.html, 1990. [Online; accessed June-2013].

[Benini *et al.*, 2000] Luca Benini, Alessandro Bogliolo, and G. De Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(3):299–316, June 2000.

[Carrasco *et al.*, 2013] Rodrigo A Carrasco, Garud Iyengar, and Cliff Stein. Single machine scheduling with job-dependent convex cost and arbitrary precedence constraints. *Operations Research Letters*, 41(5):436–441, September 2013.

[Chang *et al.*, 2011] Hyunseok Chang, Murali Kodialam, Ramana Rao Kompella, T. V. Lakshman, Myungjin Lee, and Sarit Mukherjee. Scheduling in mapreduce-like systems for fast completion time. *2011 Proceedings IEEE INFOCOM*, pages 3074–3082, April 2011.

[Chekuri and Khanna, 2004] Chandra Chekuri and Sanjeev Khanna. 11. Approximation Algorithms for Minimizing Average Weighted Completion Time. In *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, pages 1–30. 2004.

[Chekuri *et al.*, 2001] Chandra Chekuri, R. Motwani, B. Natarajan, and Cliff Stein. Approximation Techniques for Average Completion Time Scheduling. *SIAM Journal on Computing*, 31(1):146, 2001.

[Chen *et al.*, 2005] J.J. Chen, T.W. Kuo, and H.I. Lu. Power-saving scheduling for weakly dynamic voltage scaling devices. *Algorithms and Data Structures*, pages 338–349, 2005.

[Cheng *et al.*, 1998] T. C. Edwin Cheng, Adam Janiak, and Mikhail Y. Kovalyov. Bicriterion single machine scheduling with resource dependent processing times. *SIAM Journal on Optimization*, 8(2):617–630, 1998.

[Cheng *et al.*, 2001] T. C. Edwin Cheng, Adam Janiak, and Mikhail Y. Kovalyov. Single machine batch scheduling with resource dependent setup and processing times. *European Journal of Operational*, 135(1):177–183, November 2001.

[Cheung and Shmoys, 2011] Maurice Cheung and D. Shmoys. A Primal-Dual Approximation Algorithm for Min-Sum Single-Machine Scheduling Problems. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 135–146, 2011.

[Comscore, 2013] Comscore. Comscore May 2013 Ranking. Online at http://goo.gl/KLDHV, 2013.

[Cormen *et al.*, 2001] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[Crauwels *et al.*, 1998] HAJ Crauwels, C.N. Potts, and Luk N. Van Wassenhove. Local search heuristics for the single machine total weighted tardiness scheduling problem. *INFORMS Journal on ...*, 10(3):341–350, 1998.

[Croce *et al.*, 1995] Federico Della Croce, Wlodzimierz Szwarc, Roberto Tadei, Paolo Baracco, and Raffaele di Tullio. Minimizing the weighted sum of quadratic completion times on a single machine. *Naval Research Logistics*, 42(8):1263–1270, December 1995.

[Daniels and Sarin, 1989] Richard L. Daniels and Rakesh K. Sarin. Single machine scheduling with controllable processing times and number of jobs tardy. *Operations Research*, 37(6):981–984, 1989.

[Daniels, 1990] Richard L. Daniels. A multi-objective approach to resource allocation in single machine scheduling. *European journal of operational research*, 48:226–241, 1990.

[DOE, 2011] DOE. Department of Energy website. Online at http://goo.gl/BEFdZ, 2011.

[Duffuaa *et al.*, 1999] S Duffuaa, S O Duffuaa, A Raouf, and J D Campbell. *Planning and control of maintenance systems: modeling and analysis*. John Wiley & Sons Inc, 1999.

[Goemans *et al.*, 2002] Michel X. Goemans, Maurice Queyranne, Andreas S. Schulz, Martin Skutella, and Yaoguang Wang. Single Machine Scheduling with Release Dates. *SIAM Journal on Discrete Mathematics*, 15(2):165, 2002.

[Goemans, 1997] Michel X. Goemans. Improved approximation algorthims for scheduling with release dates. *ACM-SIAM symposium on Discrete algorithms*, pages 591–598, 1997.

[Google, 2009] Google. Google Datacentre Webpage. Online at http://goo.gl/44nDs, 2009.

[Graham *et al.*, 1979] R.L. Graham, Eugene L. Lawler, Jan Karel Lenstra, and Alexander H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Discrete optimization*, 5:287–326, 1979.

[Gupta and Sen, 1984] S. K. Gupta and Tapan Sen. On the Single Machine Scheduling Problem with Quadratic Penalty Function of Completion Times: An Improved Branching Procedure. *Management Science*, 30(5):644–647, 1984.

[Gurobi Optimization Inc., 2012] Gurobi Optimization Inc. Gurobi Optimizer Reference Manual. Online at http://www.gurobi.com, 2012.

[Hall and Posner, 2001] NG Hall and ME Posner. Generating experimental data for computational testing with machine scheduling applications. *Operations Research*, 49(7):854–865, 2001.

[Hall *et al.*, 1996] Leslie A Hall, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: off-line and on-line algorithms. In *Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms*, SODA '96, pages 142–151, Philadelphia, PA, USA, August 1996. Society for Industrial and Applied Mathematics.

[Hall *et al.*, 1997] 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. *Mathematics of Operations Research*, 22(3):513–544, August 1997.

[Höhn and Jacobs, 2012] W Höhn and Tobias Jacobs. An experimental and analytical study of order constraints for single machine scheduling with quadratic cost. *Proc. of the 14th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 103–117, 2012.

[Hwang and Wu, 1997] C.H. Hwang and A.C.H. Wu. A predictive system shutdown method for energy saving of event-driven computation. In *Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design*, pages 28–32. IEEE Computer Society, December 1997.

[Irani and Pruhs, 2005] Sandy Irani and Kirk R. Pruhs. Algorithmic problems in power management. *ACM SIGACT News*, 36(2):63, June 2005.

[Irani *et al.*, 2003] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Online strategies for dynamic power management in systems with multiple power-saving states. *ACM Transactions on Embedded Computing Systems (TECS)*, 2(3):346, August 2003.

[Irani *et al.*, 2007] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Algorithms for power savings. *ACM Transactions on Algorithms*, 3(4):41, November 2007.

[Janiak and Kovalyov, 1996] Adam Janiak and Mikhail Y. Kovalyov. Single machine scheduling subject to deadlines and resource dependent processing times. *European Journal of Operational Research*, 2217(96), 1996.

[Janiak, 1987] Adam Janiak. One-machine scheduling with allocation of continuously-divisible resource and with no precedence constraints. *Kybernetika*, 23(4), 1987.

[Janiak, 1991] Adam Janiak. Single machine scheduling problem with a common deadline and resource dependent release dates. *European Journal of Operational Research*, 53:317–325, 1991.

[Jawor, 2005] Wojciech Jawor. Three dozen papers on online algorithms. *ACM SIGACT News*, 36(1):71–85, 2005.

[Kalyanasundaram and Pruhs, 2000] Bala Kalyanasundaram and Kirk R. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, July 2000.

[Karlin *et al.*, 1994] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki. Competitive randomized algorithms for nonuniform problems. *Algorithmica*, 11(6):542–571, June 1994.

[Kaspi and Shabtay, 2006] Moshe Kaspi and Dvir Shabtay. A bicriterion approach to time/cost trade-offs in scheduling with convex resource-dependent job processing times and release dates. *Computers & Operations Research*, 33(10):3015–3033, October 2006.

[Kwon and Kim, 2005] Woo-Cheol; Kwon and Taewhan Kim. Optimal voltage allocation techniques for dynamically variable voltage processors. *ACM Transactions on Embedded Computing*, 4(1):211–230, 2005.

[Lawler and Moore, 1969] Eugene L. Lawler and J. Michael Moore. A Functional Equation and its Application to Resource Allocation and Sequencing Problems. *Management Science*, 16(1):77–84, September 1969.

[Lawler, 1977] Eugene L. Lawler. A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. In *Studies in integer programming (Proc. Workshop, Bonn, 1975)*, volume 1, pages 331–342, Bonn, 1977.

[McNaughton, 1959] R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 4(1956):1–12, 1959.

[Mondal and Sen, 2000] Sakib A. Mondal and Anup K. Sen. An improved precedence rule for single machine sequencing problems with quadratic penalty. *European Journal of Operational Research*, 125(2):425–428, September 2000.

[Monma *et al.*, 1990] Clyde L. Monma, Alexander Schrijver, Michael J. Todd, and Victor K. Wei. Convex resource allocation problems on directed acyclic graphs: duality, complexity, special cases, and extensions. *Mathematics of Operations*, 15(4):736–748, 1990.

[Moore, 1968] J. Michael Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15(1):102–109, 1968.

[Phillips *et al.*, 1998] Cynthia A. Phillips, Cliff Stein, and Joel Wein. Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82(1-2):199–223, June 1998.

[Phillips *et al.*, 2002] Cynthia A. Phillips, Cliff Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32(2):163–200, 2002.

[Pinedo, 2008] Michael Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer New York, New York, NY, 3rd edition, 2008.

[Potts and van Wassenhove, 1991] CN Potts and LN van Wassenhove. Single machine tardiness sequencing heuristics. *IIE transactions*, (May 2013):37–41, 1991.

[Pruhs *et al.*, 2004] Kirk R. Pruhs, Jiri Sgall, and Eric Torng. 15. Online Scheduling. In *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. 2004.

[Pruhs *et al.*, 2007] Kirk R. Pruhs, Rob Stee, and Patchrawat Uthaisombut. Speed Scaling of Tasks with Precedence Constraints. *Theory of Computing Systems*, 43(1):67–80, October 2007.

[Pruhs *et al.*, 2008] Kirk R. Pruhs, Patchrawat Uthaisombut, and Gerhard J. Woeginger. Getting the best response for your erg. *ACM Transactions on Algorithms*, 4(3):1–17, June 2008.

[Schaller and Valente, 2012] Jeffrey Schaller and Jorge M.S. Valente. Minimizing the weighted sum of squared tardiness on a single machine. *Computers & Operations Research*, 39(5):919–928, May 2012.

[Schild and Fredman, 1961] Albert Schild and IJ Fredman. On scheduling tasks with associated linear loss functions. *Management Science*, 7(3):280–285, 1961.

[Schild and Fredman, 1962] A Schild and I J Fredman. Scheduling tasks with deadlines and non-linear loss functions. *Management Science*, 9(1):73–81, 1962.

[Sen *et al.*, 2003] Tapan Sen, Joanne M Sulek, and Parthasarati Dileepan. Static scheduling research to minimize weighted and unweighted tardiness: A state-of-the-art survey. *International Journal of Production Economics*, 83(1):1–12, January 2003.

[Shabtay and Kaspi, 2004] Dvir Shabtay and Moshe Kaspi. Minimizing the total weighted flow time in a single machine with controllable processing times. *Computers & Operations Research*, 31(13):2279–2289, November 2004.

[Shabtay and Steiner, 2007] Dvir Shabtay and George Steiner. A survey of scheduling with controllable processing times. *Discrete Applied Mathematics*, 155(13):1643–1666, August 2007.

[Shabtay and Steiner, 2011] Dvir Shabtay and George Steiner. A bicriteria approach to minimize the total weighted number of tardy jobs with convex controllable processing times and assignable due dates. *Journal of Scheduling*, 14(5):455–469, November 2011.

[Skutella, 2006] Martin Skutella. List Scheduling in Order of α-Points on a Single Machine. In Evripidis Bampis, Klaus Jansen, and Claire Kenyon, editors, *Efficient Approximation and Online Algorithms*, volume 3484 of *Lecture Notes in Computer Science*, pages 250–291. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[Sleator and Tarjan, 1985] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, February 1985.

[Sousa and Wolsey, 1992] Jorge P. Sousa and Laurence A. Wolsey. A time indexed formulation of non-preemptive single machine scheduling problems. *Mathematical Programming*, 54(1-3):353–367, February 1992.

[Srivastava *et al.*, 1996] M.B. Srivastava, A.P. Chandrakasan, and R.W. Brodersen. Predictive system shutdown and other architectural techniques for energy efficient programmable computation. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 4(1):42–55, 1996.

[Townsend, 1978] W. Townsend. The single machine problem with quadratic penalty function of completion times: a branch-and-bound solution. *Management Science*, 24(5):530–534, 1978.

[Van Wassenhove and Baker, 1982] Luk N. Van Wassenhove and Kenneth R. Baker. A bicriterion approach to time/cost trade-offs in sequencing. *European Journal of Operational Research*, 1982.

[Vickson, 1980] R. G. Vickson. Choosing the job sequence and processing times to minimize total processing plus flow cost on a single machine. *Operations Research*, 28(5), 1980.

[Wang and Wang, 2011] Ji-Bo Wang and Ming-Zheng Wang. Single-machine scheduling to minimize total convex resource consumption with a constraint on total weighted flow time. *Computers & Operations Research*, 39(3):492–497, March 2011.

[Williams, 1985] T J Williams. *Analysis and design of hierarchical control systems: with special reference to steel plant operations*, volume 3. Elsevier, 1985.

[Williamson and Shmoys, 2011] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, New York, NY, USA, 1st edition, 2011.

[Xu *et al.*, 2011] Kailiang Xu, Zuren Feng, and Liangjun Ke. Single machine scheduling with total tardiness criterion and convex controllable processing times. *Annals of Operations Research*, 186(1):383–391, March 2011.

[Yao *et al.*, 1995] Frances Yao, Alan Demers, and Scott Shenker. A scheduling model for reduced CPU energy. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 374–382. IEEE Comput. Soc. Press, 1995.

[Yin, 2012] Wotao Yin. Gurobi Mex: A MATLAB interface for Gurobi. Online at http://convexoptimization.com/wikimization/index.php/gurobi_mex, 2012.

[Yun and Kim, 2003] H.S. Yun and Jihong Kim. On energy-optimal voltage scheduling for fixed-priority hard real-time systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 2(3):393–430, 2003.