Theoretical bounds for scheduling problems and their application to asymptotic analysis and energy consumption minimization

Redouane Elghazi

UNIVERSITÉ DE FRANCHE-COMTÉ

THESE DE DOCTORAT DE L'ETABLISSEMENT UNIVERSITE BOURGOGNE FRANCHE-COMTE

PREPAREE A L'UNIVERSITE DE FRANCHE-COMTE

Ecole doctorale n° 37

SCIENCES PHYSIQUES POUR L'INGENIEUR ET MICROTECHNIQUES

Doctorat d'Informatique

Par

Monsieur ELGHAZI Redouane

Theoretical bounds for scheduling problems and their application to asymptotic analysis and energy consumption minimization

Bornes théoriques de problèmes d'ordonnancement et leurs applications à l'analyse asymptotique et la minimisation de la consommation d'énergie

Thèse présentée et soutenue à Lyon, le 9 octobre 2023

Composition du jury :

Monsieur Olivier Beaumont	Directeur de recherche, INRIA Bordeaux Sud-Ouest	Examinateur
Madame Anne Benoit	Maîtresse de conférences, ENS de Lyon, LIP	Co-encadrante de thèse
Monsieur Louis-Claude Canon	Maître de conférences, Université de Franche-Comté	Co-directeur de thèse
Monsieur Georges Da Costa	Professeur, Université de Toulouse	Rapporteur
Monsieur Pierre-Cyrille Héam	Professeur, Université de Franche-Comté	Directeur de thèse
Madame Alix Munier-Kordon	Professeur, Sorbonne Université	Examinatrice
Monsieur Krzysztof Rzadca	Associate Professor, Université de Varsovie	Rapporteur

Acknowledgments

This section will be written in french as it mainly concerns french speaking people.

Cette thèse n'aurait pas été possible sans la présence, le soutien et l'aide de personnes qui me sont chères, que ce soit pendant la durée de mon doctorat ou plus largement au cours de mon voyage dans le monde merveilleux qu'est celui de l'informatique. Avant d'entrer dans le vif du sujet, je prendrai donc quelques lignes pour les remercier.

En premier lieu et en place d'honneur, je tiens à remercier mes parents qui ont su me soutenir dans tous mes projets et faire de moi la personne que je suis. À ma mère Zakia, j'ai encore occasionnellement besoin de tes relectures mais je rédige de mieux en mieux grâce aux notions de rigueur que tu m'as enseignées. À mon père Mjid, tu n'auras malheureusement pas eu le temps de lire ma thèse, mais je suis sûr que tu en aurais été fier : tu m'as grandement influencé dans mon parcours en sciences.

Ensuite, artisans directs des travaux de ce doctorat, je me dois de remercier mes 3 encadrants Anne, Louis-Claude, et Pierre-Cyrille qui m'ont accompagné pendant ces 3 années de labeur. Le doctorat est souvent considéré comme une période difficile, mais j'ai eu la chance d'être assez bien accompagné pour toujours garder la tête hors de l'eau et rester sur les bons rails. Merci Anne de m'avoir accompagné depuis mes premiers pas dans la recherche il y a presque 10 ans, je n'aurais pas pu espérer meilleure mentore. Merci Louis-Claude pour toutes les bonnes pratiques que tu m'as fait mettre en place, même si je n'ai pas encore pris toutes tes bonnes habitudes. Et merci Pierre-Cyrille pour toutes les ficelles que tu as pu me montrer.

Merci aussi à l'ensemble de mon jury de thèse, en particulier à Georges Da Costa et Krzysztof Rzadca qui ont relu ma thèse, mais aussi à Olivier Beaumont et Alix Munier-Kordon qui sont venus jusqu'à Lyon pour m'écouter parler d'algorithmes d'approximation pendant une heure. Merci tant pour vos retours constructifs que pour le regard bienveillant que vous avez pu porter à ma thèse. Merci aussi pour les discussions que l'on a eues sur mon avenir dans la recherche, qui me paraît parfois si incertain.

Pour refermer le pan académique, je remercie tous les professeurs qui ont su me guider en informatique, en mathématiques, ou dans des domaines que j'estime proches. Merci en particulier à Yves Robert, Éric Thierry, Marie Monier, Malgosia Fender, et Philippe Aubé, qui m'ont accompagné chacun à leur façon au cours des 15 dernières années.

Merci aux collègues que j'ai pu côtoyer au cours de ces trois années, qu'ils soient bisontins ou lyonnais. Merci pour les moult repas et verres que nous avons pu partager, tantôt pour parler de recherche, tantôt pour oublier la recherche.

Et enfin, merci à tous mes amis, notamment

Angèle, Alice, Loïs, Alizée, et Lilian avec qui j'ai passé tant de bonnes soirées,

Merwan, Émile, Colin, Enguerrand, Clément, Alexis, Lucas, et Athé avec qui j'ai passé tant de bonnes nuits,

et Paul qui arrive toujours comme une fleur pour me souhaiter un bon matin.

Contents

Chapter 1

Introduction

Over the last few decades, computing has allowed analyzing data that was previously tedious to interpret or even aggregate. However, as the ambition of projects increases, so does the amount of data and the resulting computation. For example, in biology, the *1000 Genomes Project*, which has been analyzing the human genome since its start in 2008, counts several hundreds of terabytes of data [**clarke2012**]. In physics, generating the image of a supermassive black hole in 2019 consumed 100 million CPU hours [**peckham2022**]. Finally, the emergence of machine learning and artificial intelligence in science as well as in daily life also requires more and more CPU time as both fields require pretreatment of large quantities of data. For instance, a recent study relied on machine learning to show that plants emit sounds when under stress [**khait2023**].

As these large computations cannot fit on a single computer, they are usually dispatched on computing centers as tasks to be executed. The said computing centers receive many tasks and must then assign each task to a specific machine, in a specific order. This is known as *scheduling the tasks* and is a fundamental problem in computer science. Choices made when scheduling the tasks can impact quantities such as the time taken before delivering the result of the computations, or the energy consumed by the execution of the tasks. When a scheduling problem is defined, the goal is usually to minimize or maximize one of these quantities, which is called the objective function. This is why scheduling problems are optimization problems. Although problems related to task scheduling have been widely studied [**feitelson1997**], the fundamental nature of this category of problems means that they vary as new computational challenges arise. In particular, specific new tasks can require specific scheduling algorithms to perform more efficiently.

Additionally, most problems related to task scheduling are NP-complete, as simply finding a schedule of n tasks on $m = 2$ processors that minimizes the total execution time is already an NP-complete problem, and it is even strongly NP-complete if the number of processors is arbitrary [**garey1979**]. In this work, we focus on two rising challenges: the ever growing amount of tasks to execute, and the growing concern around reducing the energy we consume.

Asymptotic concerns. When the number of tasks grows large, the usual techniques used to analyze an algorithm might not be relevant anymore. This is why we revisited the classical problem of scheduling n independent tasks with costs $w_1, \ldots, w_n$ onto m identical processors. The goal here is to minimize the total execution time, or makespan, usually denoted by $C_{\max}$. This problem is denoted $P||C_{\max}$ in Graham's notation [**graham79a**] and has been extensively studied in the literature.

One of the most usual approaches when studying an algorithm consists in deriving an approximation ratio: an algorithm $\mathcal{A}$ is said to have an approximation ratio c, or to be a c-approximation, if for any instance with optimal makespan OPT, the algorithm $\mathcal{A}$ outputs a schedule with makespan at most $c \times \text{OPT}$. This method is a very strong tool when studying an algorithm under the general worst-case angle. However, this method is not fit for the situation we study, as this value c can correspond to very specific instances, which get rare or even disappear when n becomes sufficiently large. This is why, in order to assess the performance of an algorithm, we instead studied the asymptotic value of the result given by the algorithm from a probabilistic point of view.

Specifically, in the first part, we focused on the algorithm *Longest Processing Time* (LPT), in which the longest task is greedily scheduled first. This algorithm is known to be a $(\frac{4}{3} - \frac{1}{3m})$-approximation [**graham1969**], but its empirical performance is much better than this. With large numbers of tasks, LPT even appears to be almost optimal.

In the first part of my thesis, we studied the asymptotic performance of LPT under different probability distributions, with a special focus on a distribution called the uniform integer composition. We did so from a very theoretical point of view through the derivation of stochastic asymptotic bounds, but also from a more practical point of view through the use of simulations.

Energy management. The dominant paradigm has been a constant race for computing speed, in order to always handle more and more tasks. However, the rising ecological concerns have started a shift of paradigm both for some researchers and for some in the industry [**georgiou2015**, **camus2017**, **dossantos2023**].

In this context, we have decided to study a scheduling problem aiming at minimizing the energy consumed during the execution of the tasks, instead of minimizing the makespan. The technique used to change the energy consumption is called **Dynamic Voltage and Frequency Scaling (DVFS)**, and it allows us to change the speed at which a processor executes a task. With this technique, there are many models that can yield the power of a processor, i.e., the energy it consumes per unit of time, as a function of the speed s at which the processor is running. For example, polynomials ($P(s) = \sum a_i \times s^i$) can be used and allow for a model that is very general, but can be hard to study. For some models of processors, a monomial ($P(s) = a \times s^k$, with k being a constant) can be enough to efficiently model the power. We chose a model that is a trade-off between these two models, where the power of the processor is composed of a static power, incurring a linear term, and a dynamic power, incurring a term of higher degree α. This is a classical model that has already been widely used in the literature, as for instance in [**bambagini2016**].

Now, the objective function is not the makespan anymore, but it is instead the energy consumption:

$$E = \sum_{j=1}^{m} \int_0^T P(s_j(t))dt \ ,$$

where $P(s_j(t))$ is the power consumed by a processor using the speed that was allocated to processor j at time t. This problem of energy minimization has two major differences with the makespan minimization problem: there are new variables s_j corresponding to the speed of the processors at any time, and the objective function is not a linear function of the variables anymore, since the power P is not a linear function of the speed s.

In the first energy minimization problem, we studied parallel tasks that are said to be **moldable**. It means that for each task, the scheduling algorithm must choose a number of processors that will execute the task concurrently. Executing a task with more processors means that it will end earlier, but the parallelization might incur an overhead, depending on the task. This overhead might come from communications between the processors or from the fact that some parts of the task cannot be parallelized.

In this context of energy minimization, we studied two classes of schedules that are already well-known for the makespan minimization problem: *shelf schedules* and *list schedules*. The former class consists in executing consecutive "shelves" of tasks, where a shelf is a set of at most m tasks started at the same time. The latter class consists in choosing the next task to be executed according to a priority policy (e.g., Longest Processing Time, or LPT, that was mentioned earlier).

In the second part of my thesis, we derived algorithms creating shelf and list schedules in a context of energy minimization with moldable tasks. We then studied the performance of these algorithms. We did so from a theoretical point of view through the derivation of approximation ratios, but also from a more practical point of view through the use of simulations.

Energy management in an asymptotic context. After having tackled both challenges, i.e., the ever-growing amount of tasks to execute and the growing concern around reducing the energy we consume, we studied both problems at the same time. Indeed, in real life, both problems are not disjoint, and reducing the energy consumption of a schedule is still relevant when the number of tasks n grows large.

In this context, we studied tasks that are executed on only one processor each (i.e., sequential tasks), and instead of studying LPT, we studied a newer algorithm called SLACK, which was proposed in 2020 by F. Della Croce and R. Scatamacchia [**della2020**]. This algorithm comes from the observation that if we have access to m tasks with similar execution times, then allocating each of these m tasks to a different processor should keep the m processors balanced. With this in mind, SLACK creates groups of m tasks with execution times that are as close as possible, and then sequentially allocates the groups of tasks, decreasing the imbalance between the processors as the different groups are handled.

In the third part of my thesis, we derived algorithms creating schedules in a context of energy minimization by adapting the policies of SLACK and LPT. We derived general

results to help in the theoretical study of the performance of an algorithm in terms of energy consumption. We then applied these tools to SLACK to derive asymptotic bounds and convergence. We finally studied the created algorithms from a more practical point of view through the use of simulations.

Content of this thesis

This thesis is decomposed in three parts, each corresponding to an article published during the course of my PhD with my supervisors Anne Benoit, Louis-Claude Canon, and Pierre-Cyrille Héam.

Each part provides answers to some of the previously presented challenges, i.e., studying sets of tasks with a size growing to infinity, or studying the energy consumption of a schedule. In each case, after modeling the problem, there is first a study of the problem from a theoretical point of view, and then a study of the problem from a more practical point of view.

The content of the Chapter 2 has been published at Euro-Par 2021 [**benoit2021**]. An extension of this work is under review for a publication in the Journal of Scheduling. We study the problem of the minimization of the makespan of a schedule with sequential tasks, when the amount of tasks grows to infinity, and the execution times of the tasks are sampled according to a specific distribution. More specifically, we first derive a new convergence result when the distribution is the uniform integer composition: a total work is distributed over the task costs as a composition, and each composition has the same probability of occurrence. In this case, the times are not sampled independently. Then, we perform an empirical analysis of five heuristics including LPT and SLACK, comparing the algorithms one to each other, but also to the theoretical bound when there are known results.

The content of the Chapter 3 has been published at SBAC-PAD 2021 [**benoit2021shelf**] and extended in the Journal of Parallel and Distributed Computing [**benoit2023list**]. We study the problem of the minimization of the energy consumption of a schedule with moldable tasks. More specifically, we first formalize the problem, which we call MinE-Mold, with various model variants. Then, we show that if each processor can be powered off or on independently from the others, then the problem can be solved in polynomial time, and that else, the problem is NP-complete. In the NP-complete case, we derive multiple approximation ratios for two classes of algorithms: list-based algorithms and shelf-based algorithms. We then provide an optimal dynamic programming solution of the sub-problem where all tasks are placed on a single shelf (i.e., when all tasks must begin their execution at time $t = 0$), which can be useful for shelf-based algorithms. Finally, we perform an empirical study, including three main highlights. The first highlight is a comparison of the different algorithms we designed, both in terms of energy consumed by the schedule and in terms of time complexity of the scheduling algorithm. The second highlight is the fact that for most instances, running every processor at the same fixed speed does not increase the energy consumption too much, if the speed is correctly chosen. The final highlight is an assessment of the speeds available for two existing processors, in terms of energy consumption.

The content of the Chapter 4 is to be published at Euro-Par 2023 [**benoit2023asymptotic**]. We study the problem of the minimization of either the makespan or the energy consumption of a schedule with sequential tasks, when the amount of tasks grows to infinity, and the execution times of the tasks are sampled according to a specific distribution. We focused the analysis on the algorithm SLACK. We first derived a bound related to the result of SLACK and explaining its good practical performance. We used this bound for several results: we provided convergence rates for SLACK when minimizing the makespan, and when minimizing the energy consumption. In the latter case, we also provided a result for bounding the energy consumption of an algorithm, that does not depend of the algorithm used. Finally, we ran simulations in order to compare SLACK to LPT and to the theoretical bounds provided earlier.

Chapter 2

Asymptotic optimality of LPT

When independent tasks are to be scheduled onto identical processors, the typical goal is to minimize the makespan. A simple and efficient heuristic consists in scheduling first the task with the longest processing time (LPT heuristic), and to plan its execution as soon as possible. While the performance of LPT has already been largely studied, in particular its asymptotic performance, we revisit results and propose a novel analysis for the case of tasks generated through uniform integer compositions. Also, we perform extensive simulations to empirically assess the asymptotic performance of LPT, and compare it to four other classical heuristics. The results show that the absolute error rapidly tends to zero for several distributions of task costs, including distributions studied by theoretical models, and realistic distributions coming from benchmarks.

2.1 Introduction

We revisit the classical problem of scheduling n independent tasks with costs $w_1, \ldots, w_n$ onto m identical processors. The goal is to minimize the total execution time, or *makespan*, usually denoted by $C_{\max}$. This problem, denoted $P||C_{\max}$ in Graham's notation [**graham79a**], has been extensively studied in the literature, and greedy heuristics turn out to have theoretical guarantees and to perform well in practice. In particular, we focus on the *Longest Processing Time (LPT)* heuristic, where the longest task will be scheduled first, on the processor where it can start the earliest. This heuristic is very simple and has a low complexity, while exhibiting good worst-case performance with an approximation ratio of $\frac{4}{3} - \frac{1}{3m}$ [**graham1969**], and excellent empirical performance. With a large number of tasks, LPT appears to be almost optimal.

Since the worst-case performance exhibits cases where LPT is far from the optimal, many different approaches have tried to fill the gap between this worst-case performance and the excellent practical performance. The goal is to provide performance guarantees of different kinds, for instance by studying the average-case complexity, some generic-case complexity, or convergence results.

Hence, many convergence results have been proposed in the literature. They state that LPT ends up providing an optimal solution when the number of tasks grows towards

infinity. Some of these results even provide asymptotic rates that quantify the speed with which LPT tends to optimality. These results depend on assumptions on the probability distribution of the costs of the tasks, and on the definition of distance to optimality. However, the literature lacks a definitive answer on the convergence to optimality and its rate when faced with difficult cost distributions. In particular, this work is the first to consider dependent random costs with a constraint on the minimum cost.

First, Section 2.2 synthesizes the existing contributions and their limitations. Next, we describe the five considered heuristics in Section 2.3, in particular a novel strategy, SLACK, recently proposed in [**della2020**], and the Largest Differencing Method (LDM) that provides a similar approximation ratio to LPT [**michiels2003**]. We also provide an analysis of the complexity of LPT for small instances. Then, we revisit LPT and propose an update to the already known asymptotic optimality results, both from a theoretical perspective and from an empirical one. Our main contribution is twofold:

1. We derive a new convergence (in probability) result when the distribution of task costs is generated using uniform integer compositions, hence leading to a novel probabilistic analysis of the heuristics for this problem (Section 2.4);

2. We perform a thorough empirical analysis of the five heuristics, with an extended range of settings to study particular distributions but also distributions coming from real applications (Section 2.5).

Finally, conclusions and future work directions are discussed in Section 2.6.

2.2 Related work

Theoretical studies

There are several theoretical works studying the rate of convergence of LPT. [**coffman1982**] analyze the average performance of LPT under the assumption that costs are uniformly distributed in the interval $(0, 1]$. They show that the ratio between the expected makespan obtained with LPT and the expected optimal one with preemption is bounded by $O(1 + \frac{m^2}{n^2})$, where m is the number of processors and n is the number of tasks.

[**frenk1986**] bound the *absolute error* (i.e., the difference between the achieved makespan and the optimal one) of LPT using order statistics of the processing times when the cost distribution has a cumulative distribution function of the form $F(x) = x^a$ with $0 < a < \infty$. The results also stand when this constraint is relaxed into $F(x) = \Theta(x^a)$. They prove that the absolute error goes to 0 with speed $O\left((\frac{\log\log(n)}{n})^{\frac{1}{a}}\right)$ as the number of tasks n grows. For higher moments, of order q, a similar technique gives a speed of $O\left((\frac{1}{n})^{\frac{a}{q}}\right)$.

[**frenk1987**] also study uniform machines ($Q||C_{\max}$) in the more general case where costs follow a distribution with finite moment and the cumulative distribution function is strictly increasing in a neighbourhood of 0. They show that LPT is asymptotically optimal almost surely in terms of absolute error. When it is the second moment that

is finite instead, they show that LPT is asymptotically optimal in expectation. For the more specific cases where the costs follow either a uniform distribution or a negative exponential distribution, they provide additional convergence rates.

Another theoretical study is done by [**loulou1984**], providing a comparison between LPT and a less sophisticated heuristic, RLP (Random List Processing), also called LS (List Scheduling) in this thesis. This heuristic is simpler than LPT because the jobs are considered in an arbitrary order instead of a sorted order. These algorithms are studied under the assumption that the costs are independent and identically distributed (i.i.d.) random variables with finite first moment. Under this assumption, the absolute error of RLP with at least three processors and LPT are both stochastically bounded by a finite random variable. The author also proves that the absolute error of LPT converges in distribution to optimality with rate $O(1/n^{1-\epsilon})$.

[**coffman1988**] list various results and techniques that are useful for the study of the problems of scheduling and bin packing. They consider both theoretical optimal results, and heuristic algorithm results. LPT is one of the algorithms they study, in terms of both relative error (LPT/OPT) and absolute error (LPT − OPT). They also reuse the specific probability distribution used by [**frenk1986**] of the form $F(x) = x^a$, with $0 < a < \infty$. They present a heuristic adapted from a set-partitioning problem with a better convergence on this distribution.

[**piersma1996**] consider the $R||C_{\max}$ problem (with unrelated machines), and they propose an LP relaxation of the problem, followed by a Lagrange relaxation. Assuming that the processing times are i.i.d. random vectors of $[0, 1]^m$, they prove that $\frac{1}{n}$OPT converges almost surely to a value θ that they give (it depends on the Lagrange relaxation). Using a previous convergence result [**frenk1986**], they infer that the makespan of LPT also converges a.s. to $n\theta$.

[**dempster1983**] consider an objective function also depending on the machine cost, and they propose a heuristic in two steps, where they first choose the machines to be bought with knowledge of the distribution of the jobs, and then schedule the jobs on the machines that were bought in the first step. For identical machines, assuming that the processing times are i.i.d. random variables with finite second moment, they prove that the relative error of their heuristic converges to 0 in expectation and probability when the number of jobs goes to infinity. For uniform machines, they need more assumptions to reach results.

Summary

Table 2.1 summarizes the main results that are known about LPT.

Beyond LPT

Even though LPT has interesting properties in terms of convergence, other heuristics have been designed for the multiprocessor scheduling problem. For independent tasks and makespan minimization, the problem is actually close to a bin-packing problem,

	Problem	Distribution	Studied quantity	Convergence/rate
[**coffman1982**]	$P\|\|C_{\max}$	$\mathcal{U}(0,1)$	$E[\text{LPT}]/E[\text{OPT}^*]$	$1+O(m^2/n^2)$
[**frenk1986**]	$P\|\|C_{\max}$	$F(x)=x^a$, $0<a<\infty$	LPT $-$ OPT	$O((\log\log(n)/n)^{\frac{1}{a}})$ almost surely (a.s.)
[**frenk1986**]	$P\|\|C_{\max}$	as above	$E[(\text{LPT}-\text{OPT})^q]$	$O((1/n)^{\frac{a}{q}})$
[**frenk1987**]	$Q\|\|C_{\max}$	finite 1st moment	LPT $-$ OPT	a.s.
[**frenk1987**]	$Q\|\|C_{\max}$	finite 2nd moment	LPT $-$ OPT	in expectation
[**frenk1987**]	$Q\|\|C_{\max}$	$\mathcal{U}(0,1)$ or $Exp(\lambda)$	LPT $-$ OPT	$O(\log n/n)$ a.s.
[**frenk1987**]	$Q\|\|C_{\max}$	$\mathcal{U}(0,1)$	$E[\text{LPT}]-E[\text{OPT}]$	$O(m^2/n)$
[**loulou1984**]	$P\|\|C_{\max}$	finite 1st moment	LPT $-$ OPT	bounding finite RV
[**loulou1984**]	$P\|\|C_{\max}$	$\mathcal{U}(0,1)$	LPT $-$ OPT	$O(1/n^{1-\epsilon})$ in dist.
[**coffman1988**]	$P\|\|C_{\max}$	$\mathcal{U}(0,1)$	$E[\text{LPT}-\text{OPT}]$	$O(m/(n+1))$
[**piersma1996**]	$R\|\|C_{\max}$	$\mathcal{U}(0,1)$	OPT	$n\theta$ a.s.

Table 2.1: For each main result, the problem may consider uniform processors (P) or processors with speeds (Q or R). A result on the absolute difference is stronger than on the ratio. OPT is the optimal makespan, whereas OPT* is the optimal makespan with preemption.

where one would like to create m bins of same size. Hence, the MULTIFIT heuristic [**coffman1978**] builds on techniques used in bin-packing, and it provides an improved worst-case bound.

Then, a COMBINE heuristic was proposed [**lee1988**], combining MULTIFIT and LPT to get the best of these two heuristics. Another alternative, LISTFIT, was proposed in [**gupta2001**], still with the goal to minimize the makespan on identical machines.

The Largest Differencing Method (LDM) of [**karmarkar1982**] has been proven to be a $(\frac{4}{3} - \frac{1}{3m})$-approximation [**michiels2003**] similarly to LPT, while outperforming LPT and MULTIFIT from an average-case perspective [**graham1969**, **coffman1978**]. LDM is also asymptotically optimal for the problem $2||C_{\max}$ when the tasks have uniform costs in $[0, 1]$, with LDM $-$ OPT converging at rate $n^{-\Theta(\log n)}$ [**yakir1996**].

More recently, [**della2020**] revisit LPT to propose another heuristic, SLACK, by splitting the sorted tasks in tuples of m consecutive tasks (recall that m is the number of processors), and then sorting tuples by non-increasing order of the difference between the largest and smallest task in the tuple. A list-scheduling strategy is then applied with tasks sorted in this order. Moreover, LPT last step is enhanced to reach a better worst-case approximation ratio.

Empirical studies

An empirical comparison of LISTFIT with MULTIFIT, COMBINE and LPT is proposed in [**gupta2001**]. Several parameters are varied, in particular the number of machines, number of jobs, and the minimum and maximum values of a uniform distribution for processing times. No other distribution is considered. LISTFIT turns out to be robust and returns better makespan values than previous heuristics.

[**behera2012**] consider the three heuristics MULTIFIT, COMBINE and LISTFIT, and propose a comprehensive performance evaluation. While LISTFIT outperforms the two other heuristics, this comes at a price of an increased time complexity. They do not consider instances with more than 300 tasks, and no comparison with LPT is done.

An empirical evaluation of LPT was proposed in [**laha2017**], showing that LPT consumes less computational time than the competitors (MULTIFIT, COMBINE, LISTFIT), but returns schedules with higher makespan values. However, here again, there is no study of the convergence, and no comparison of LPT with other simpler algorithms.

Finally, an evaluation of SLACK is done in [**della2020**]: this variant of LPT turns out to be much better than LPT on benchmark literature instances, and it remains competitive with the COMBINE heuristic that is more costly and more difficult to implement.

Beyond independent tasks

While we have been focusing so far on independent tasks, there have also been some empirical analysis of list scheduling for general directed acyclic graphs (DAGs), i.e., with dependencies. For instance, [**cooper1998**] evaluate various list schedulers on benchmark codes, pointing out cases where a basic list-scheduling algorithm works well, and where

more sophisticated approaches are helpful. In this chapter, we focus on independent tasks to study the convergence of LPT and other heuristics.

2.3 Algorithms and complexity

We first review the algorithms considered in this work in Section 2.3.1, before discussing the complexity of LPT for small instances in Section 2.3.2.

2.3.1 Algorithms

We consider a total of five algorithms, most of them being list-scheduling algorithms. A list-scheduling algorithm orders the tasks in some way, and then greedily assigns tasks to the processor that has the lowest current finishing time (or makespan). Hence, tasks are always started as soon as possible, and for independent tasks, there is no idle time in the schedule.

We first recall four list-scheduling algorithms, which differ in the way they order the tasks:

- LS: List Scheduling is the basic list-scheduling algorithm that does not order the tasks, but rather considers them in an arbitrary order. The time complexity of LS is $O(n \log m)$.
- LPT: Largest Processing Time orders the tasks from the largest to the smallest. The time complexity of LPT is $O(n \log n)$.
- MD: Median Discriminated is an attempt to find an intermediate solution between LPT and LS. The tasks are not completely sorted, but the median of the execution times is computed so that the first $\frac{n}{2}$ processed tasks are larger than the median, while the next $\frac{n}{2}$ are smaller. The time complexity of MD is $O(n \log m)$.
- SLACK: as defined by [**della2020**], it makes packs of m tasks and defines for each of these packs the slack, which is the difference between the largest and the smallest task of the pack. The packs are then sorted from the largest to the smallest slack, and the tasks are sorted according to the order of the packs. The time complexity of SLACK is $O(n \log n)$.

We also consider one algorithm that is not a list-scheduling algorithm: the Largest Differencing Method (LDM) as defined in [**karmarkar1982**]. This algorithm creates partial solutions and merges them greedily:

- The algorithm starts by creating n partial solutions $(s_i)_i \leq n$ with solution i only having task T_i on one processor;
- At each step, the algorithm selects the two partial solutions with highest difference between the busiest processor and the least busy one;

- These two partial solutions are merged by matching the busiest processors of the first instance with the least busy ones of the second instance;
- At the end, after $n-1$ steps, there is only one remaining solution, which is a complete solution.

The time complexity of LDM is $O(nm \log m)$. An example of execution of LDM is given in Fig. 2.1, with seven tasks and three processors.

2.3.2 Optimality for small instances

In this section, we show that LPT returns an optimal solution for small instances. This can be partially explained by the following well-known property.

Lemma 1. *If there exists an optimal solution with at most two tasks per processor, then* LPT *is optimal.*

Proof. Let $n = 2m - h$, with h a non-negative integer. As there are at most two tasks per processor in the optimal solution, there are h tasks alone on their processor. We can swap these tasks with the h largest tasks without increasing $C_{\max}$. Since LPT starts by scheduling these h largest tasks, they will also each be alone on a dedicated processor, and we now consider the $n - h = 2(m - h)$ remaining tasks to be scheduled on $m - h$ processors.

There exists an optimal solution with exactly two tasks on each of the remaining $m-h$ processors. By exchange argument, if two tasks $t_i \leq t_j$ are on the same processor, and two other tasks $t_k \leq t_l$ are on another one, then we either have $t_i \leq t_k \leq t_l \leq t_j$ or $t_k \leq t_i \leq t_j \leq t_l$ (if this is not possible, we can exchange the tasks without increasing the makespan). From that, we get that the processor executing the longest of the $2(m-h)$ tasks will also execute the smallest of these tasks. We then use the same argument for the second largest task, and so on and so forth, to find that the solution given by LPT is an optimal solution. □

Note that this property becomes false as soon as we have more than two tasks on a specific processor. With three tasks on a processor, we can already reach the worst case of LPT [**graham1969**] ($\frac{\text{LPT}}{\text{OPT}} = \frac{4}{3} - \frac{1}{3m}$) with the family of instances $(I_m)_{m>1}$ such that the instance I_m is defined as follows:

- There are $m+1$ processors;
- There are a total of $n = 2(m+1)$ tasks:
  - $2(m-1)$ tasks with costs $w_{2i-1} = w_{2i} = m + i$, $i \in [1, m-1]$ (costs ranging from $m+1$ to $2m-1$, each duplicated);
  - Three tasks with cost m each;
  - One task with cost $3m$.

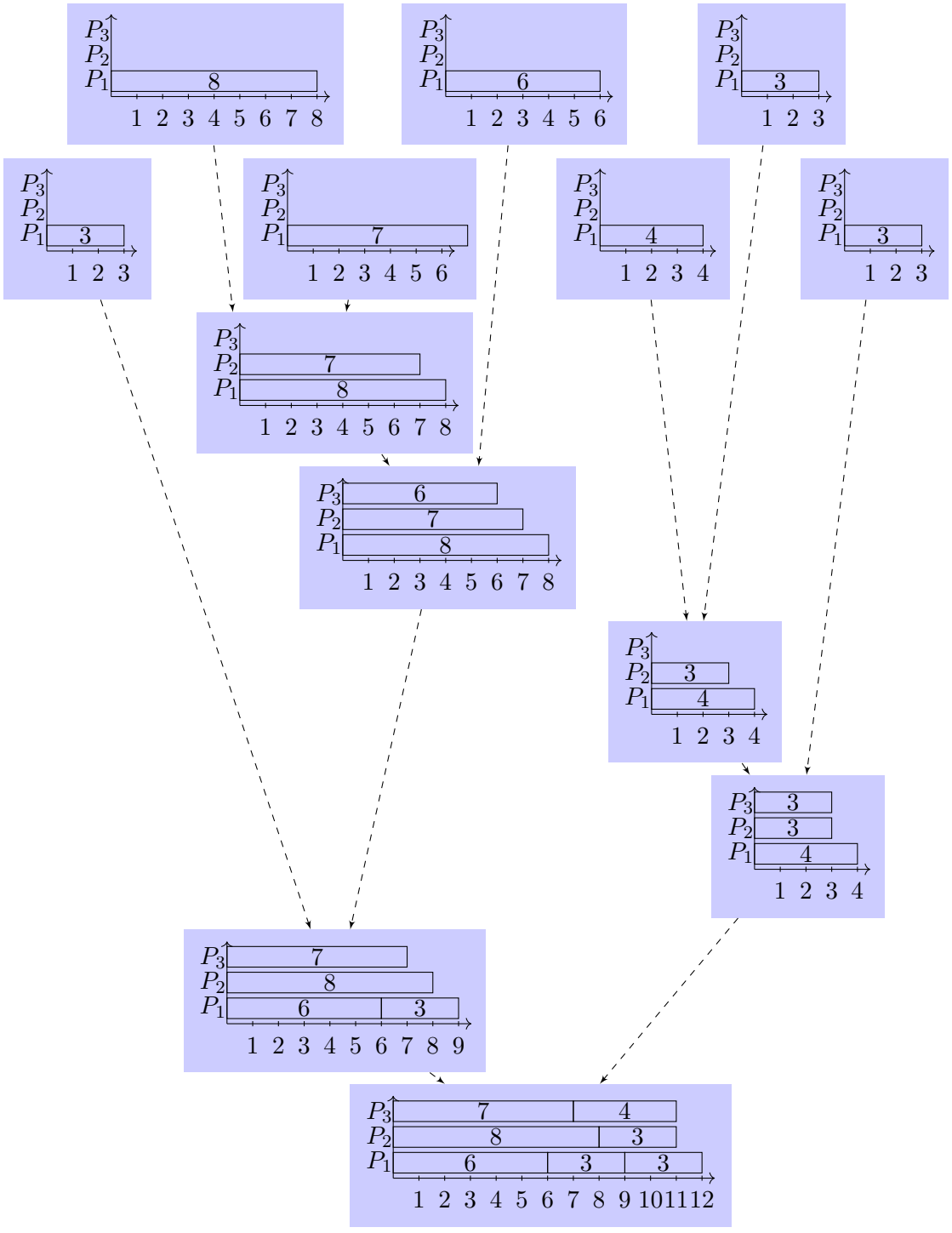
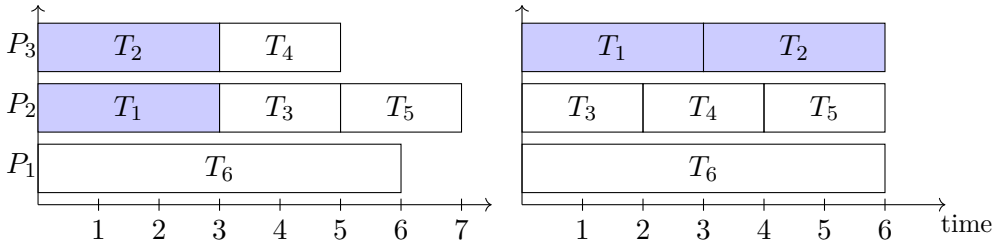


Figure 2.1: Example of execution of LDM on an instance with seven tasks with costs (3,3,3,4,6,7,8) and with three processors. There are seven initial solutions, that are then merged (one step per line). The final solution has (7,4) on the first processor, (8,3) on the second, and (6,3,3) on the last, with a makespan of 12 reached on processor 3. In this case, LDM is optimal.



(a) LPT schedule **(b) Optimal schedule**

Figure 2.2: Counter example with $m = 2$, 3 tasks on a processor.

For instance I_m, LPT achieves a makespan of $4m-1$ by putting the task with cost $3m$ alone, and then allocating two tasks per remaining processor, reaching a cost of $3m - 1$ on each remaining processor. Finally, the remaining task with cost m is allocated to a processor that already has a total cost of $3m - 1$, reaching a makespan of $4m - 1$.

For the same instance I_m, a makespan of $3m$ can be achieved by putting the task with cost $3m$ alone, the three tasks with cost m together, and pairing the $2(m - 1)$ remaining tasks in pairs with cost $3m$ each.

Thus, the approximation ratio reached by LPT for this family of instances is $\frac{\text{LPT}}{\text{OPT}} = \frac{4}{3} - \frac{1}{3m}$.

This is illustrated for $m = 2$ in Fig. 2.2, where the $2(m - 1)$ tasks are the two identical tasks highlighted in blue. Since they are larger than the tasks of size m, LPT is scheduling these first, while they should have been paired together on a same processor in an optimal solution, retrieving the well-known worst case of $\frac{7}{6}$.

2.4 Convergence results for integer compositions

In this section, we derive new convergence results for the algorithms that were described in Section 2.3.1. These results apply when the distribution of task costs is generated following an integer composition method. In contrast to related work where the number of tasks n is known beforehand, this consists in considering that the total amount of work W is fixed (costs are thus dependent random variables). We detail how tasks are generated among possible decompositions of this work (Section 2.4.1). We finally perform the probabilistic analysis in two different settings, depending whether the minimum cost of tasks is one (Section 2.4.2) or greater (Section 2.4.3).

The proofs of the results in this section are mainly based on combinatorics techniques. The reader is referred to [**Flajolet2009**] for more information.

2.4.1 Tasks random generation

A W-composition is a finite sequence $w_1, \ldots, w_n$ of strictly positive integers such that $w_1 + \ldots + w_n = W$.

Let $\mathbb{D}_W$ be the uniform distribution over W-compositions and $\mathbb{D}_{W,w_{\min}}$ the uniform distribution over W-compositions satisfying for each i, $w_i \geq w_{\min}$. In particular, $\mathbb{D}_{W,1} = \mathbb{D}_W$. For instance, $\mathbb{D}_4$ is the uniform distribution over the eight elements $(1,1,1,1)$, $(1,1,2)$, $(1,2,1)$, $(1,3)$, $(2,1,1)$, $(2,2)$, $(3,1)$, (4); $\mathbb{D}_{4,2}$ is the uniform distribution over $(2,2)$ and (4). Note that for $\mathbb{D}_4$, the probability that $w_1 = 1$ is $1/2$ and the probability that $w_1 = 3$ is $1/8$.

In practice, random generation is performed using the recursive method [**DBLP:journals/tcs/Flaj**

For a list L of task costs, we denote by $\mathrm{LPT}(L,m)$ the makespan $C_{\max}$ returned by LPT on m machines. We define as well $\mathrm{LS}(L,m)$, $\mathrm{MD}(L,m)$, $\mathrm{SLACK}(L,m)$, and $\mathrm{LDM}(L,m)$ for the other heuristics. The optimal (minimum) $C_{\max}$ that can be obtained by any algorithm is similarly denoted $\mathrm{OPT}(L,m)$.

2.4.2 Probabilistic analysis for D_W

Ratio for $\mathbb{D}_W$

In this setting, we know the total workload W, but the number of tasks n is not fixed and there is no minimum task cost. Let $L[W] = (w_1, \ldots, w_n)$ be a sequence of positive integers such that $\sum_{i=1}^{n} w_i = W$, hence a W-composition.

According to [**graham1969**],

$$\begin{aligned}\frac{\mathrm{LS}(L[W],m)}{\mathrm{OPT}(L[W],m)} &\leq 1 + (m-1)\frac{w_{\max}}{\sum_{i=1}^{n} w_i} \\ &= 1 + (m-1)\frac{w_{\max}}{W},\end{aligned}$$

where $w_{\max} = \max_{1 \leq i \leq n}\{w_i\}$.

Following [**Flajolet2009**], for $\mathbb{D}_W$ and for any y,

$$\mathbb{P}\left(w_{\max} \geq 2\log_2 W + y\right) = O\left(\frac{e^{-2y}}{W}\right). \tag{2.1}$$

Since by definition of OPT, $\mathrm{OPT}(L[W],m) \leq \mathrm{LS}(L[W],m)$, for any fixed m,

$$\mathbb{P}\left(\frac{\mathrm{LS}(\mathbb{D}_W,m)}{\mathrm{OPT}(\mathbb{D}_W,m)} \leq 1 + 2(m-1)\frac{\log_2(W)}{W}\right) \underset{W\to+\infty}{\longrightarrow} 1.$$

It is also known, see [**Flajolet2009**], that for the distribution $\mathbb{D}_W$, $E[w_{\max}] \sim \log_2 W$. By linearity of expectations, the following result holds:

$$E\left[\frac{\mathrm{LS}(\mathbb{D}_W, m)}{\mathrm{OPT}(\mathbb{D}_W, m)}\right] \underset{W\to+\infty}{\longrightarrow} 1.$$

The results also hold for LPT, MD and SLACK, which are particular list-scheduling heuristics. It also holds for LDM, as the algorithm has as invariant that the idle time per processor is no more than the largest task of the schedule.

Absolute error for $\mathbb{D}_W$

The *absolute error* of a heuristic is the difference between its result and the optimal result. A first obvious upper bound is that $LS(L,m) - OPT(L,m) \leq w_{\max}$ (for any set of tasks L), and previous results on $w_{\max}$ can be used to bound the error (but not to prove that it tends to 0). Furthermore, we prove the following theorem:

Theorem 1. *Algorithms* LPT, SLACK, *and* LDM *are optimal for* $\mathbb{D}_W$*, with probability* $1 - O\left(\frac{1}{W}\right)$*. Under the same conditions and with the same probability,* MD *is near optimal. For any fixed* m*, for* L *generated according to* $\mathbb{D}_W$*,*

$$\mathbb{P}(\mathrm{LPT}(L,m) = \mathrm{OPT}(L,m)) = 1 - O\left(\frac{1}{W}\right),$$

$$\mathbb{P}(\mathrm{SLACK}(L,m) = \mathrm{OPT}(L,m)) = 1 - O\left(\frac{1}{W}\right),$$

$$\mathbb{P}(\mathrm{LDM}(L,m) = \mathrm{OPT}(L,m)) = 1 - O\left(\frac{1}{W}\right),$$

and

$$\mathbb{P}(\mathrm{MD}(L,m) \leq \mathrm{OPT}(L,m) + 1) = 1 - O\left(\frac{1}{W}\right).$$

To prove this theorem, let us first prove two lemmas. Let α_W be the random variable counting the number of w_i's equal to 1 in $\mathbb{D}_W$.

Lemma 2. *For* $W \geq 3$*,*

$$E[\alpha_W] = \frac{W+2}{4} \quad \textit{and} \quad \mathrm{Var}[\alpha_W] = \frac{5}{16}(W+1).$$

Proof. There are 2^{W-1} compositions of W. The expected number of parts n is $\frac{W+1}{2}$. The number of 1-parts in a composition of W is given by the ordinary generating function $C(z,u) = \frac{1}{1-(\frac{z}{1-z}+(u-1)z)} = \frac{1-z}{1-2z-z(1-z)(u-1)}$ (see [**Flajolet2009**]).

Moreover, we have [**Flajolet2009**]:

$$E[\alpha_W] = \frac{[z^W]\partial_u C(z,u)_{|u=1}}{[z^W]C(z,1)} = \frac{[z^W]\partial_u C(z,u)_{|u=1}}{2^{W-1}}, \tag{2.2}$$

and

$$E[\alpha_W^2] = \frac{[z^W]\partial_u^2 C(z,u)_{|u=1}}{[z^W]C(z,1)} + \frac{[z^W]\partial_u C(z,u)_{|u=1}}{[z^W]C(z,1)}. \tag{2.3}$$

Also, $\partial_u C(z,u) = \frac{z(1-z)^2}{(1-2z-(u-1)z(1-z))^2}$. Therefore $\partial_u C(z,u)_{|u=1} = \frac{z(1-z)^2}{(1-2z)^2} = \frac{z}{4} - \frac{1}{4} + \frac{1}{8(1-2z)} + \frac{1}{8(2z-1)^2}$. Using Equation (2.2), we obtain for $W \geq 2$,

$$E[\alpha_W] = \frac{W+2}{4}.$$

Now, $\partial_u^2 C(z,u) = \frac{2z^2(1-z)^3}{(1-2z-z(1-z)(u-1))^3}$ and $\partial_u^2 C(z,u)_{|u=1} = \frac{2z^2(1-z)^3}{(1-2z)^3} = \frac{z^2}{4} - \frac{3z}{8} - \frac{1}{8(1-2z)} + \frac{1}{16(1-2z)^2} + \frac{1}{16(1-2z)^3}$.

It follows that for $W \geq 3$,

$$E[\alpha_W^2] = \frac{W(W+1)}{16} + \frac{W+1}{8} - \frac{1}{4} + \frac{W+2}{4}.$$

It follows that:
$\mathrm{Var}[\alpha_W] = E[\alpha_W^2] - E[\alpha_W]^2 = \frac{5}{16}(W+1)$. □

Lemma 3. *Let A_W be the event $\alpha_W > \frac{W}{8}$ and B_W be the event $w_{\max} \leq 2\log_2 W$. For $\mathbb{D}_W$, we have:*

$$\mathbb{P}(A_W \cap B_W) = 1 - O\left(\frac{1}{W}\right).$$

Proof. We have:

$$\begin{aligned}
\mathbb{P}\left(\alpha_W \leq \frac{W}{8}\right) &= \mathbb{P}\left(\alpha_W \leq \frac{W+2}{4} - \frac{W+4}{8}\right) \\
&\leq \mathbb{P}\left(\alpha_W \leq \frac{W+2}{4} - \frac{W+4}{8}\right) \\
&+ \mathbb{P}\left(\alpha_W \geq \frac{W+2}{4} + \frac{W+4}{8}\right) \\
&= \mathbb{P}\left(\left|\alpha_W - \frac{W+2}{4}\right| \geq \frac{W+4}{8}\right).
\end{aligned}$$

Now, using Lemma 2 and Chebyshev's inequality, we obtain:

$$\mathbb{P}\left(\alpha_W \leq \frac{W}{8}\right) \leq \frac{5}{16}\frac{W+1}{(\frac{W+4}{8})^2} = 20\frac{W+1}{(W+4)^2} = O(\frac{1}{W}).$$

It follows that $\mathbb{P}(\overline{A_W}) = O(\frac{1}{W})$. Furthermore, according to Equation (2.1), $\mathbb{P}(\overline{B_W}) = O(\frac{1}{W})$. Finally,

$$\begin{aligned}
\mathbb{P}(A_W \cap B_W) &= 1 - \mathbb{P}(\overline{A_W} \cup \overline{B_W}) \\
&\leq 1 - \mathbb{P}(\overline{A_W}) - \mathbb{P}(\overline{B_W}) \\
&= 1 - O\left(\frac{1}{W}\right),
\end{aligned}$$

which concludes the proof. □

We are now ready to prove Theorem 1.

Proof. By Lemma 3, one has with probability $1 - O(\frac{1}{W})$, $\alpha_W = |\{i \mid w_i = 1\}| > \frac{W}{8}$ and $w_{\max} \leq 2\log_2 W$.

Assume that $L[W]$ satisfies these two properties, and let δ be the maximum difference of loads between two processors when there remains $\lceil \frac{W}{8} \rceil$ tasks to scheduled (scheduling with LPT). One has $\delta \leq w_{\max} \leq 2\log_2 W$.

Now, the remaining tasks to be scheduled are unitary. Since the function mapping x to $x - 16(m-1)\log_2 x$ is strictly increasing for $x \geq 1$ and tends to infinity when $x \to +\infty$, there exists an integer W_0 such that, for every $W \geq W_0$, $\frac{W}{8} \geq 16(m-1)\log_2 W$.

Consequently, the remaining unitary tasks will be optimally scheduled, proving the theorem for LPT.

For the SLACK algorithm, the proof is quite similar. By Lemma 3, one has with probability $1 - O(\frac{1}{W})$, $\alpha_W > \frac{W}{8}$ and $w_{\max} \leq 2\log_2 W$. In this case, there are at least $\frac{W}{8(m+1)}$ m-tuples appearing in the SLACK algorithm with a null slack and composed of tasks of cost 1. Moreover each m-tuple has a maximal $2\log_2 W - 1$ slack (when the tuple contains a maximal and a minimal w_i). Therefore, for W large enough, the scheduling of the $\frac{W}{8(m+1)}$ tuples of unitary tasks fulfills any difference between the current processing times of the machines. Note that we may still have some non-unitary tasks to schedule, but since tuples are sorted by non-increasing slack, they would also be organized in m-tuples with null stack.

For the LDM algorithm, the proof also uses Lemma 3. Recall that LDM creates partial solutions, and at each step, the two partial solutions with the highest slack are merged. By Lemma 3, one has with probability $1 - O(\frac{1}{W})$, $\alpha_W > \frac{W}{8}$ (i.e., there are initially at least $\frac{W}{8}$ unitary tasks) and $w_{\max} \leq 2\log_2 W$ (i.e., the highest slack is at most $2\log_2 W$ at any time of the algorithm). Assume for the rest of the proof that this is the case.

At some point in the execution of LDM, there is at most one main partial solution with a slack strictly greater than 1, and there are at least $\frac{W}{8}$ partial solutions consisting of a single unitary task. We will now prove that from this situation, we get a final solution with a slack of at most 1.

Let $T_{\max}$ be the makespan of the main partial solution, $T_{\min}$ the execution time of the least loaded processors in this partial solution, and $m_{\min}$ the number of such processors, i.e., the number of processors that have the lowest load. Let I be the following bound on the idle time of the main partial solution:

$$I = (m-1)T_{\max} - m_{\min}T_{\min} - (m - 1 - m_{\min})(T_{\min} + 1).$$

When merging a partial solution of slack at least 2 with a single unitary task, the quantity I decreases, as either $T_{\max} - T_{\min}$ decreases, or it does not change and $m_{\min}$ decreases. When merging a partial solution of slack at least 2 with a partial solution of slack 1, I does not increase, as $T_{\max} - T_{\min}$ does not increase, and the only way for $m_{\min}$ to increase is for $T_{\min}$ to increase while $T_{\max}$ does not.

From the bound on $w_{\max}$, we know the starting value of I is at most $(m-1)w_{\max} = (m-1)2\log_2 W$ and we do at least $\frac{W}{8}$ merges with unitary tasks. For W large enough,

it either means that the main partial solution gets a slack of at most 1 at some point, or I becomes negative. As I cannot be negative, it means that at some point the main partial solution gets a slack that is less or equal to 1.

A solution with a slack of 0 or 1 is optimal, so this proves the theorem for LDM.

It remains to prove a similar result for MD but up to 1 to the optimal. Let $\rho_W = |\{i \mid w_i = 2\}|$. The bivariate generating function associate to ρ_W is

$$G(z,u) = \frac{1-z}{1-2z+(u-1)(1-z)z^2}.$$

Since $\partial_u G(z,u)_{|u=1} = \frac{(1-z)^2z^2}{(1-2z)^2} = z\partial_u A(z,u)_{|u=1}$, we have, $[z^W]\partial_u G(z,u)_{|u=1} = [z^{W-1}]\partial_u A(z,u)_{|u=1}$. It follows that, for $W \geq 2$, $E[\rho_W] = \frac{W+1}{8}$.

Similarly $\partial_u^2 G(z,u)_{|u=1} = z^2\partial_u^2 A(z,u)$, providing that, for $W \geq 3$, $E[\rho_W^2] = \frac{(W-2)(W-1)}{64} + \frac{W-1}{32} - \frac{1}{4} + \frac{W+1}{8}$. It follows that $\mathrm{Var}[\rho_W] = \frac{9W-25}{64}$.

Using Chebyshev's inequality as in Lemma 2, one can prove that $\mathbb{P}(\rho_W \leq \frac{W}{16}) = O(\frac{1}{W})$. One can also prove similarly that $\mathbb{P}(\alpha_W \leq \frac{11W}{64}) = O(\frac{1}{W})$.

Consequently, with probability $1 - O(\frac{1}{W})$ one has $\alpha_W > \frac{11W}{64}$ and $\rho_W > \frac{W}{16}$ and $w_{\max} \leq 2\log_2 W$. Assume for the rest of the proof that this is the case.

Let n be the number of tasks. There are $n - \alpha_W - \rho_W$ that are of cost greater than or equal to 3. Therefore, $n - \alpha_W - \rho_W \leq \frac{W-\alpha_W-2\rho_W}{3}$. Consequently $n - 2\alpha_W - 2\rho_W \leq \frac{W-4\alpha_W-5\rho_W}{3} \leq \frac{W}{3}(1 - \frac{11}{16} - \frac{5}{16}) \leq 0$. Since $\alpha_W + \rho_W \geq \frac{n}{2}$, the task of medium value has either cost 1 or 2. If it is 1, then MD is optimal (with the same arguments as for LPT). Otherwise, MD is up to 1 to the optimal. □

Theorem 1 can be reformulated in a convergence in probability result:

Corollary 1. *For every $\varepsilon > 0$, for the distributions $\mathbb{D}_W$,*

$$\lim_{W\to+\infty} \mathbb{P}(|\mathrm{LPT}(L,m) - \mathrm{OPT}(L,m)| \geq \varepsilon) = 0,$$

$$\lim_{W\to+\infty} \mathbb{P}(|\mathrm{SLACK}(L,m) - \mathrm{OPT}(L,m)| \geq \varepsilon) = 0,$$

$$\lim_{W\to+\infty} \mathbb{P}(|\mathrm{LDM}(L,m) - \mathrm{OPT}(L,m)| \geq \varepsilon) = 0,$$

and

$$\lim_{W\to+\infty} \mathbb{P}(|\mathrm{MD}(L,m) - \mathrm{OPT}(L,m)| \geq 1+\varepsilon) = 0.$$

Proof. $\mathbb{P}(|\mathrm{LPT}(L,m) - \mathrm{OPT}(L,m)| \geq \varepsilon) = \mathbb{P}(\mathrm{LPT}(L,m) - \mathrm{OPT}(L,m) \geq \varepsilon) \leq \mathbb{P}(\mathrm{LPT}(L,m) - \mathrm{OPT}(L,m) > 0) = 1 - \mathbb{P}(\mathrm{LPT}(L,m) - \mathrm{OPT}(L,m) = 0) = O\left(\frac{1}{W}\right)$. The proof is similar for SLACK, LDM, and MD. □

2.4.3 Analysis for $\mathbb{D}_{W,w_{\min}}$

Let $\min_{1\leq i\leq n}\{w_i\} = w_{\min} \geq 2$. Let $\alpha_{W,w_{\min}}$ be the number of w_i's equal to $w_{\min}$ in a decomposition $(w_1,\ldots,w_n)$ satisfying $\sum_{i=1}^n w_i = W$ and for all $1 \leq i \leq n$, $w_i \geq w_{\min}$. The random variable $\alpha_{W,w_{\min}}$ is studied for the $\mathbb{D}_{W,w_{\min}}$ distribution. Let also $\gamma_{w_{\min},k}$ be the number of w_i's greater than or equal to k (with $k \geq w_{\min}$).

Theorem 2. *Let m be a fixed number of machines. If L is generated according to $\mathbb{D}_{W,w_{\min}}$, we have:*
$\mathbb{P}(|\mathrm{LPT}(L,m) - \mathrm{OPT}(L,m)| \leq w_{\min}) \underset{W\to+\infty}{\longrightarrow} 1$,
$\mathbb{P}(|\mathrm{SLACK}(L,m) - \mathrm{OPT}(L,m)| \leq w_{\min}) \underset{W\to+\infty}{\longrightarrow} 1$,
$\mathbb{P}(|\mathrm{LDM}(L,m) - \mathrm{OPT}(L,m)| \leq w_{\min}) \underset{W\to+\infty}{\longrightarrow} 1$.

The proof is based on two lemmas.

Lemma 4. *There exists a constant $\beta > 0$ such that $E[\alpha_{W,w_{\min}}] \underset{W\to+\infty}{\sim} \beta(n+1)$. Moreover,* $\mathrm{Var}[\alpha_{W,w_{\min}}] = o(W^2)$.

Proof. The ordinary generating function for compositions $(w_1,\ldots,w_n)$ satisfying $\sum_{i=1}^n w_i = W$ and for all $1 \leq i \leq n$, $w_i \geq w_{\min}$ is:

$$A(z) = \frac{1}{1-\frac{z^{w_{\min}}}{1-z}} = \frac{1-z}{1-z-z^{w_{\min}}}.$$

Let $Q(z) = 1-z-z^w_{\min}$. The polynomial Q is strictly decreasing on $\mathbb{R}^+$ and $Q(0) = 1$ and $Q(1) = 1$. Therefore, Q has a unique real root σ satisfying $0 < \sigma < 1$. Consequently, there exists a polynomial P with real coefficients such that $Q(z) = (\sigma - z)P(z)$. Moreover, since $\lim_{z\to\sigma^-} Q(z) = 0^+$ and since Q has no multiple roots, $P(\sigma) > 0$.

It follows that $A(z) \underset{z\to\sigma}{\sim} \frac{1-\sigma}{(\sigma-z)P(\sigma)}$. Using transfer results [**Flajolet2009**], we obtain:

$$[z^W]\ A(z) \underset{W\to+\infty}{\sim} \frac{1-\sigma}{\sigma P(\sigma)}\sigma^{-W}. \tag{2.4}$$

The bivariate ordinary generating function for the number decompositions $(w_1,\ldots,w_n)$ satisfying $\sum_{i=1}^n w_i = W$ and for every i, $w_i \geq w_{\min}$, and counting the number of w_i's equal to $w_{\min}$ is

$$\begin{aligned} A(z,u) &= \frac{1}{1-(\frac{z^{w_{\min}}}{1-z} - (u-1)z^{w_{\min}})} \\ &= \frac{1-z}{1-z-z^{w_{\min}}+(1-z)(u-1)z^{w_{\min}}}. \end{aligned}$$

It follows that

$$\partial_u A(z,u) = \frac{(1-z)^2 z^{w_{\min}}}{(1-z-z^{w_{\min}}+(1-z)(u-1)z^{w_{\min}})^2}$$

and

$$\begin{aligned}\partial_u A(z,u)_{|u=1} &= \frac{(1-z)^2 z^{w_{\min}}}{(1-z-z^{w_{\min}})^2} \\ &= \frac{(1-z)^2 z^{w_{\min}}}{(\sigma-z)^2 P(z)^2} \underset{z\to\sigma}{\sim} \frac{(1-\sigma)^2\sigma^w_{\min}}{(\sigma-z)^2 P(\sigma)^2}.\end{aligned}$$

Therefore, using [**Flajolet2009**] again,

$$[z^W]\ \partial_u A(z,u)_{|u=1} \underset{W\to+\infty}{\sim} \frac{(1-\sigma)^2\sigma^w_{\min}}{\sigma^2 P(\sigma)^2}\sigma^{-W}(W+1). \tag{2.5}$$

Consequently, using [**Flajolet2009**] and Equations (2.4) and (2.5),

$$\begin{aligned}E[\alpha_{W,w_{\min}}] &\underset{W\to+\infty}{\sim} \frac{(1-\sigma)^2\sigma^w_{\min}}{\sigma^2 P(\sigma)^2}\frac{\sigma P(\sigma)}{1-\sigma}(W+1) \\ &= \frac{(1-\sigma)\sigma^{w_{\min}-1}}{P(\sigma)}(W+1) = \beta(W+1),\end{aligned}$$

with $\beta = \frac{(1-\sigma)\sigma^{w_{\min}-1}}{P(\sigma)}$.

It remains to prove the result for the variance. One has

$$\partial_u^2 A(z,u)_{|u=1} = \frac{2(1-z)^3 z^{2r}}{(\sigma-z)^3 P(z)^3} \underset{z\to\sigma}{\sim} \frac{2(1-\sigma)^3\sigma^{2r}}{(\sigma-z)^3 P(\sigma)^3}.$$

Consequently,

$$[z^W]\ \partial_u^2 A(z,u)_{|u=1} \tag{2.6}$$

$$\underset{W\to+\infty}{\sim} \frac{2(1-\sigma)^3\sigma^{2r}}{\sigma^3 P(\sigma)^3}\frac{(W+1)(W+2)}{2}\sigma^{-W}. \tag{2.7}$$

According to [**Flajolet2009**] and using Equation (2.4),

$$\begin{aligned}E[\alpha^2_{W,w_{\min}}] &\underset{W\to+\infty}{\sim} \frac{2(1-\sigma)^3\sigma^{2r}}{\sigma^3 P(\sigma)^3}\frac{\sigma P(\sigma)}{1-\sigma}\frac{(W+1)(W+2)}{2} \\ &\quad + \beta(W+1) \\ &\underset{W\to+\infty}{\sim} \beta^2(W+1)(W+2).\end{aligned}$$

We have proved that $\frac{E[\alpha_{W,w_{\min}}]}{W} \underset{W\to+\infty}{\longrightarrow} \beta$ and $\frac{E[\alpha^2_{W,w_{\min}}]}{W^2} \underset{W\to+\infty}{\longrightarrow} \beta^2$. It follows that

$$\frac{\mathrm{Var}[\alpha_{W,w_{\min}}]}{W^2} = \frac{E[\alpha^2_{W,w_{\min}}] - E[\alpha_{W,w_{\min}}]^2}{W^2} \underset{W\to+\infty}{\longrightarrow} 0,$$

which concludes the proof. □

Lemma 5. *One has, for the $D_{W,w_{\min}}$ distribution,*

$$\mathbb{P}(w_{\max} \geq \log^2_{1/\sigma}(W)) \underset{W\to+\infty}{\longrightarrow} 0,$$

where σ is the unique real root of $1-z-z^k=0$ ($0<\sigma<1$).

Proof. Let $B(z,u)$ be the bivariate ordinary generating function of $\mathbb{D}_{W,w_{\min}}$, with parameter the number $\gamma_{w_{\min},k}$.

One has $B(z,u)=\frac{1}{1-(\frac{z^{w_{\min}}}{1-z}+(u-1)\frac{z^k}{1-z})}=\frac{1-z}{1-z-z^{w_{\min}}+(u-1)z^k}$. The proof lies on Markov inequality. We will point out an upper bound of $E[\gamma_{w_{\min},k}]$. One has

$$E[\gamma_{w_{\min},k}]=\frac{[z^W]\ \partial_u B(z,u)_{|u=1}}{[z^W]B(z,1)}. \tag{2.8}$$

Now, $\partial_u B(z,u)_{|u=1}=\frac{(1-z)z^k}{(1-z-z^{w_{\min}})^2}$. Since $\frac{1-z}{(1-z-z^{w_{\min}})^2}\underset{\sigma}{\sim}\frac{1-\sigma}{(z-\sigma)^2P(\sigma)^2}$, using [**Flajolet2009**] one has $[z^W]\ \frac{1-z}{(1-z-z^{w_{\min}})^2}\underset{W\to+\infty}{\sim}\frac{1-\sigma}{\sigma^2P(\sigma)^2}\sigma^{-W}(W+1)$, that can be reformulated into $[z^W]\ \frac{1-z}{(1-z-z^{w_{\min}})^2}=\frac{1-\sigma}{\sigma^2P(\sigma)^2}\sigma^{-W}(W+1)(1+\varepsilon(W))$, with $\varepsilon(W)\to 0$ when $W\to+\infty$. Note too that $\varepsilon(W)$ depends on $w_{\min}$ but not on k. It provides

$$[z^W]\ \partial_u B(z,u)_{|u=1}=[z^{W-k}]\ \frac{1-z}{(1-z-z^{w_{\min}})^2} \tag{2.9}$$

$$=\frac{1-\sigma}{\sigma^2P(\sigma)^2}\sigma^{-W+k}(W-k+1)(1+\varepsilon(W-k)). \tag{2.10}$$

Similarly, $[z^W]\ B(z,1)_{|u=1}=[z^W]\ \frac{1-z}{(1-z-z^{w_{\min}})^2}=\frac{1-\sigma}{\sigma P(\sigma)}\sigma^{-W}(1+\varepsilon'(W))$, with $\varepsilon'(W)\to 0$ when $W\to+\infty$. Note that $\varepsilon'(W)$ depends on $w_{\min}$ but not on k. Therefore, and combining (2.8) and (2.10), we obtain

$$E[\gamma_{w_{\min},k}]=\frac{\sigma^k(W+1-k)}{\sigma P(\sigma)}\frac{1+\varepsilon(W-k)}{1+\varepsilon'(W)}. \tag{2.11}$$

With $k=\log^2_{1/\sigma}W$, Equation (2.11) becomes

$$\begin{aligned}E[\gamma_{w_{\min},k}]&=\frac{\sigma^{\log^2_{1/\sigma}W}(W+1-\log^2_{1/\sigma}W)}{\sigma P(\sigma)}\\&\quad\times\frac{1+\varepsilon(W-\log^2_{1/\sigma}W)}{1+\varepsilon'(W)}\\&\leq\frac{W^{1-\log_{1/\sigma}W}}{\sigma P(\sigma)}\frac{1+\varepsilon(W-\log^2_{1/\sigma}W)}{1+\varepsilon'(W)}.\end{aligned}$$

Since $0<\sigma<1$, $W^{1-\log_{1/\sigma}W}\underset{W\to+\infty}{\longrightarrow}0$.

To finish, using Markov inequality,

$$\begin{aligned}\mathbb{P}(w_{\max} \geq \log^2_{1/\sigma}(W))) &= \mathbb{P}(\gamma_{w_{\min},\log^2_{1/\sigma}(W)} \geq 1) \\ &\leq E[\gamma_{w_{\min},\log^2_{1/\sigma}(W)}],\end{aligned}$$

proving the lemma. □

We are now ready to prove Theorem 2.

Proof. We only provide a sketch of the proof, which is similar to the one of Theorem 1. Using Lemma 4 and Lemma 5, almost surely there is a linear number (relatively to W) of tasks of cost $w_{\min}$ and the maximum cost of a task is bounded by $\log^2_{1/\sigma}$. Therefore, applying LPT will provide a maximum load C such that $C - W/m \leq w_{\min}$. The inequality $\mathrm{OPT} \geq W/m$ concludes the proof. The proofs for SLACK and LDM follow as for Theorem 1. □

We do not have yet any theoretical results for MD for $\mathbb{D}_{W,w_{\min}}$, but experimental results explored in Section 2.5 are encouraging.

As for Theorem 1, one can deduce the following corollary from Theorem 2 (with a similar proof).

Corollary 2. *For every $\varepsilon > 0$, every $w_{\min} \geq 2$, for the distributions $\mathbb{D}_{W,w_{\min}}$,*

$$\lim_{W\to+\infty} \mathbb{P}(|\mathrm{LPT}(L,m) - \mathrm{OPT}(L,m)| \geq w_{\min} + \varepsilon) = 0,$$

$$\lim_{W\to+\infty} \mathbb{P}(|\mathrm{SLACK}(L,m) - \mathrm{OPT}(L,m)| \geq w_{\min} + \varepsilon) = 0,$$

and

$$\lim_{W\to+\infty} \mathbb{P}(|\mathrm{LDM}(L,m) - \mathrm{OPT}(L,m)| \geq w_{\min} + \varepsilon) = 0.$$

2.5 Empirical study

The objective of this section is threefold: first, evaluate the tightness of the convergence rate proposed in [**frenk1986**] (Section 2.5.2); then, assess the performance of the five heuristics when generating costs with the integer composition approach (Section 2.5.3); finally, quantifying the convergence for realistic instance (Section 2.5.4). We first detail the experimental setting in Section 2.5.1. All the algorithms were implemented in Python 3, and the code is available on figshare[1].

[1] https://doi.org/10.6084/m9.figshare.14755296

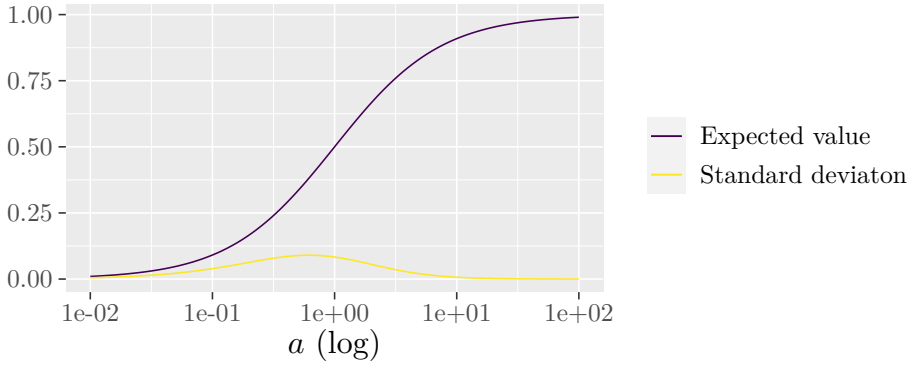


Figure 2.3: Expected value and standard deviation of a random variable with cumulative distribution function $F(x) = x^a$ as a function of a with $0 < a < \infty$.

2.5.1 Experimental setting

Synthetic instances

We consider two kinds of synthetic instances:

1. i.i.d. execution times with cumulative distribution function $F(x) = x^a$ for some $a > 0$. This distribution has an expected value of $\frac{a}{a+1}$ and a variance of $\frac{a}{(a+1)^2 \cdot (a+2)}$. These values can be seen as a function of a in Fig. 2.3. Note that for $a = 1$, this is a uniform distribution $\mathcal{U}(0, 1)$.

2. The integer composition distribution considered in Section 2.4, that is to say a uniform distribution on all possible ways to decompose a total amount of work into integer values.

Realistic instances

We also compare the five algorithms of Section 2.3.1 using real logs from the Parallel Workloads Archive, described in [**feitelson2014**] and available at https://www.cs.huji.ac.il/labs/parallel/workload/. More specifically, we took the instances called KIT ForHLR II[2] with 114 355 tasks and NASA Ames iPSC/860[3] with 18 239 tasks. The profiles of the task costs in these instances are presented in Fig. 2.4.

In order to also get instances for which the number of tasks n could change, we build new instances from these two instances. In the new instances, the tasks are i.i.d. random variables with an empirical cumulative distribution function that is computed from the distribution of the two original instances.

[2] https://www.cs.huji.ac.il/labs/parallel/workload/l_kit_fh2/index.html
[3] https://www.cs.huji.ac.il/labs/parallel/workload/l_nasa_ipsc/index.html

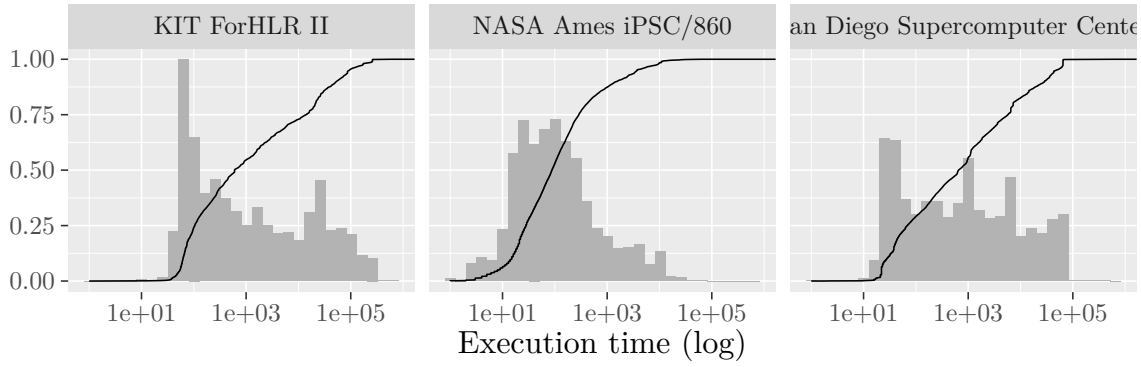


Figure 2.4: Empirical cumulative distributions and histograms of task costs for the KIT ForHLR II, NASA Ames iPSC/860 and Supercomputer Center (SDS) instances.

Optimality transform

When studying the absolute error of an algorithm, we consider the difference of its makespan to the optimal one to measure the convergence when the number of tasks n goes to infinity. The optimal makespan is computationally hard to get, so as a first approach, we can take a lower bound instead of the actual optimal value. However, there is a risk of actually measuring the quality of the lower bound instead of the quality of the algorithm.

To address this problem, we transform the instances so that we know the optimal makespan. This transformation is described as follows:

- We take an instance with n tasks;
- We perform a random List Scheduling on this instance;
- From this schedule, we add a total of at most $m - 1$ tasks so that all of the processors finish at the same time;
- We randomize the order of the tasks to avoid adding a bias to the heuristics;
- We end up with an instance with at most $n + m - 1$ tasks such that the optimal makespan equals the sum of the execution times divided by the number of processors ($\text{OPT} = \frac{W}{m}$).

As we are interested in the asymptotic behavior of the algorithms, m is small compared to n, and we expect this transformation to alter the task distribution only marginally. Even if there are few such tasks, we still analyze how much they differ from the original tasks of the instance, through an empirical study.

The distribution of the added tasks can be found in Fig. 2.5, along with the cumulative distribution function of the tasks originally in the instance (i.e., $F(x) = x^a$). We

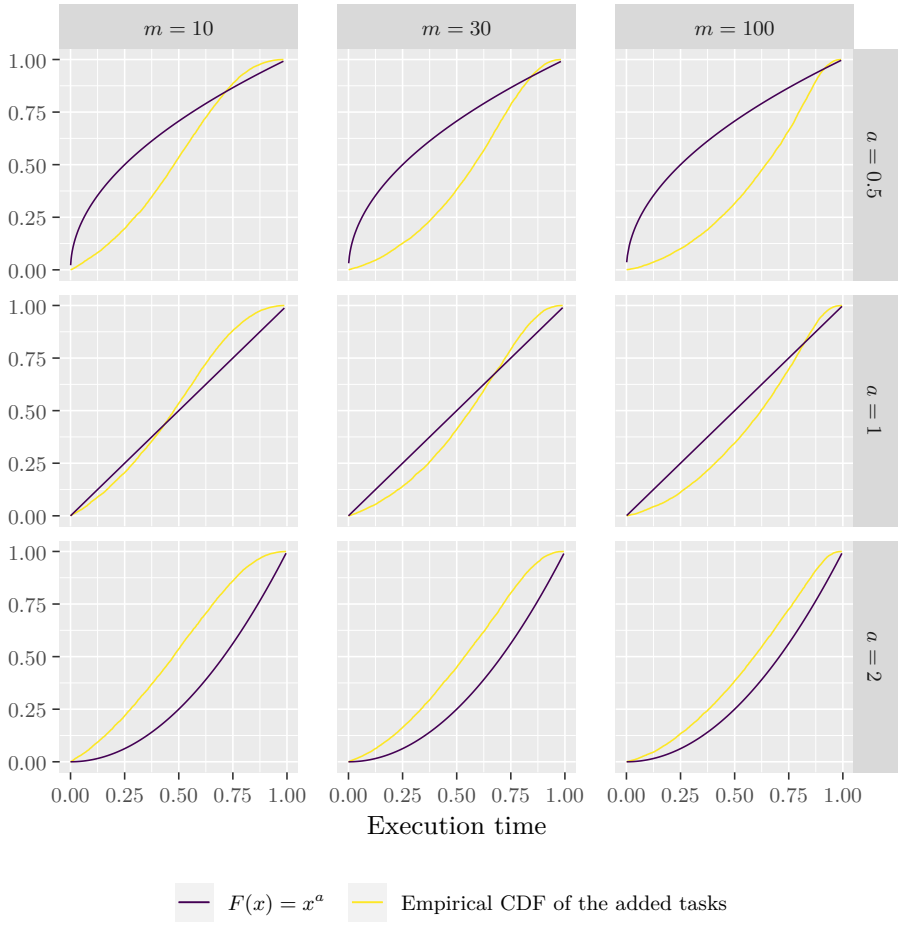


Figure 2.5: Cumulative distribution function for the added tasks compared to the cumulative distribution function of the original tasks, on instances of 500 tasks for $F(x) = x^a$.

can see that for the three instances where $a = 0.5$, the two distributions do not resemble each other much, with Kolmogorov-Smirnov statistics[4] between 0.3 and 0.42. For the instances with $a = 1$ (i.e., the uniform distribution), the added tasks resemble the original tasks much more, with Kolmogorov-Smirnov statistics between 0.12 and 0.16, and for the instances with $a = 2$, the similarity between the added tasks and the original tasks varies a lot, with Kolmogorov-Smirnov statistics between 0.16 and 0.31.

2.5.2 Rate tightness

We experimentally verify the bound given in [**frenk1986**]: if the tasks are independent and have cumulative distribution function $F(x) = x^a$ with $a > 0$, then the absolute error is a $O((\frac{\log\log(n)}{n})^{\frac{1}{a}})$ almost surely.

Fig. 2.6 depicts the absolute error of LPT and related heuristics (LS, MD, SLACK, and LDM) for different values of n. The instance contains $n - m + 1$ costs generated with the considered distribution and is then completed with the optimality transform. Moreover, we plot $C \times (\frac{\log\log(n)}{n})^{\frac{1}{a}}$, where C is the lowest constant such that all of LPT values are under the bound.

We can see that the bound seems to be rather tight for LPT, which confirms that the convergence rate of [**frenk1986**] is strong. Also, we can see that the absolute error of SLACK and LDM seems to converge to 0 at a similar rate as LPT, but with a lower multiplicative constant. Their performances are very similar in most case, except for both a high number of processors and a high value of a, in which case SLACK performs better than LDM. On the other side, the absolute errors of LS and MD do not seem to converge to 0 at all, but MD performs significantly better than LS.

2.5.3 Uniform integer compositions

Some experiments have been performed for the distributions described in Section 2.4: a total workload W is fixed as well as a fixed number m of machines. Then, the list of task costs is uniformly picked among all the possible lists for the distribution $\mathbb{D}_W$; and among all the possible lists with a minimum cost $w_{\min}$ for the distribution $\mathbb{D}_{W,w_{\min}}$.

For $\mathbb{D}_W$, an instance has been generated for all W from 10 to 9999, for $m = 10$, $m = 30$, and $m = 100$. Instances are not transformed to avoid changing the total work W. Thus, we compare the makespan obtained by the heuristics to the lower bound on the optimal value OPT: $\max(\lceil \frac{W}{m} \rceil, w_{\max})$. In all cases (about 30 000), LPT and SLACK always reach this bound, which indicates that they are both optimal and the bound is tight with these instances. LDM was found to be suboptimal in only 2 cases, that is to say 0.0067% of the cases. Results for LS and MD are reported in Table 2.2. The average absolute error for MD is 0.35 for this experiment with a standard deviation of 0.6. Moreover MD is optimal in 67.6% of the samples and up to 1 from the optimum in 98% of the samples. LS is optimal in 3.3% of the cases and the average error is 3.75 (s.d. 2.15).

[4]Presented by [**berger2014**], the Kolmogorov-Smirnov statistic of two distributions measures how

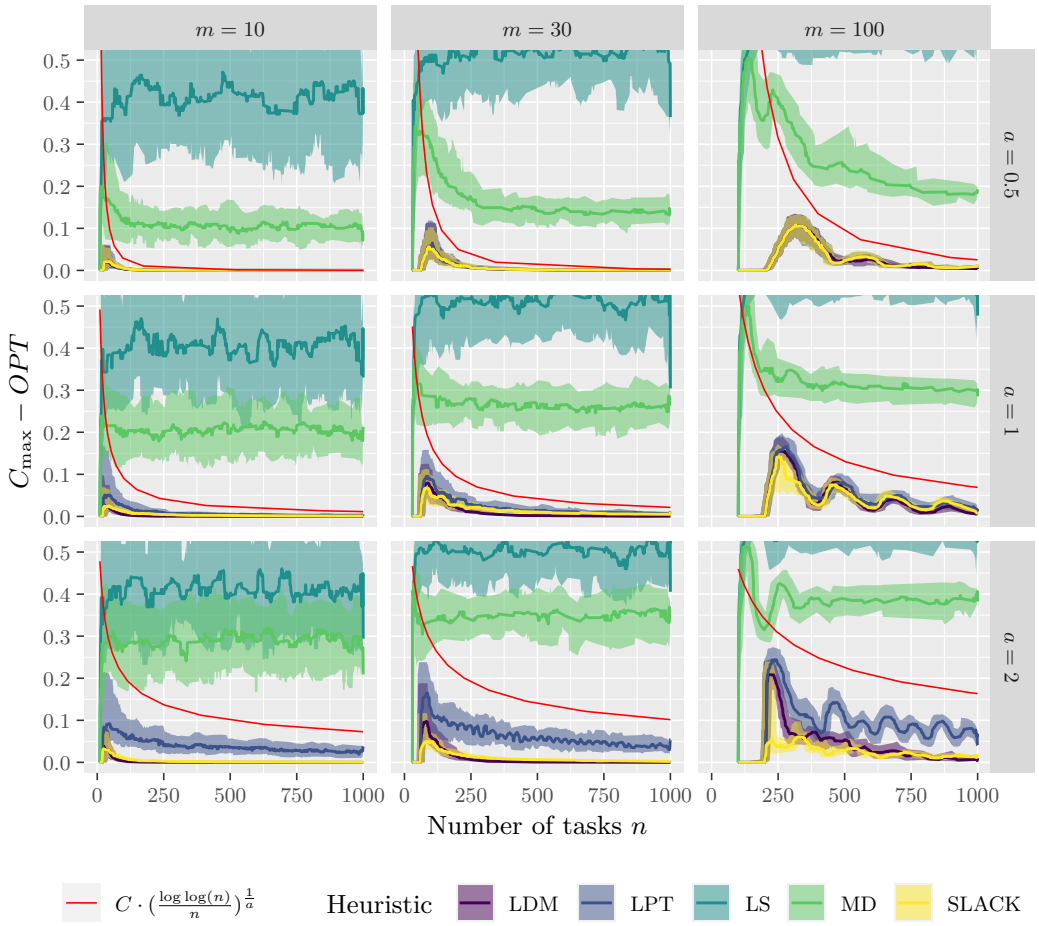


Figure 2.6: Absolute error with a distribution of the form $F(x) = x^a$ with $a > 0$ (instances are transformed to obtain OPT). Smoothed lines are obtained by using a rolling median with 45 values (each value is set to the median of the 22 values on the left, the 22 on the right and the current one). The ribbons represent the rolling 0.1- and 0.9-quantiles.

abs. err.	LS	MD	abs. err.	LS	MD
0	3.4	67.7	6	9.1	0.04
1	10.7	30.3	7	5.0	$<$ 0.01
2	17.0	0.96	8	2.7	0
3	18.1	0.58	9	1.5	0
4	17.3	0.28	10	0.7	0
5	13.7	0.08	>10	0.6	0

Table 2.2: Distribution in percentages of the absolute errors observed for LS and MD with W from 10 to 9999 and $m \in \{10, 30, 100\}$.

$w_{\min}$	LPT	LS	MD	SLACK	LDM
3	2 – 0.93 – 0.70	21 – 2.54 – 0.91	5 – 1.48 – 0.56	2 – 0.61 – 0.60	2 – 0.57 – 0.57
5	4 – 1.84 – 1.23	26 – 4.21 – 1.48	9 – 2.69 – 0.87	5 – 1.11 – 0.86	4 – 1.07 – 0.82
7	6 – 2.71 – 1.80	48 – 6.02 – 2.08	13 – 3.65 – 1.15	6 – 1.60 – 1.17	6 – 1.56 – 1.11
10	9 – 3.99 – 2.65	37 – 8.42 – 3.12	15 – 5.23 – 1.80	11 – 2.32 – 1.62	9 – 2.29 – 1.57

Table 2.3: Results on the difference between the $C_{\max}$ computed by the heuristics and a lower bound of the optimal makespan OPT. Each line is related to different $D_{W,w_{\min}}$. The first number is the maximum difference observed for all the samples, the second one is the average difference, and the last one is the standard deviation of this difference. Each value is obtained with W from 10 to 9999 and for $m \in \{10, 30, 100\}$.

Similar tests have been done for $\mathbb{D}_{W,w_{\min}}$ with $W \in \{10, \ldots, 9999\}$, $w_{\min} \in \{3, 5, 7, 10\}$ and $m \in \{10, 30, 100\}$ (see Table 2.3). We now focus on the difference δ between $C_{\max}$ and the lower bound. In each case, the maximal value of δ is reported, as well as its average and standard deviation. Note that for each sample, both SLACK and LDM ensure that $\delta < w_{\min}$, while SLACK and MD ensures it in more than 99% of cases. The LS heuristic is less effective since for $w_{\min} = 3$, only 42% of the samples satisfy $\delta < w_{\min}$; 49% for $w_{\min} = 5$, 53% for $w_{\min} = 7$ and 58% for $w_{\min} = 10$. Results for the SLACK and LDM heuristics are very close, and better than LPT. Over the about 120 000 samples, SLACK is strictly better than LPT 68202 times, while LPT is strictly better than SLACK only 1797 times. The difference can be up to 8 units of time. LDM is strictly better than SLACK 3352 times, by up to 8 units of time, while SLACK is better than LDM only 362, by up to only 2 units of time. LDM performs better than both SLACK and LPT, but the cost for this performance is a higher execution time.

2.5.4 Realistic workloads

In Fig. 2.7, we present experiments similar to those with synthetic instances in Section 2.5.2, but with the realistic instances.

As we can see when comparing LS and MD, treating the $\frac{n}{2}$ largest tasks first only

close the two distributions are (the lower the statistic is, the closer the distributions are).

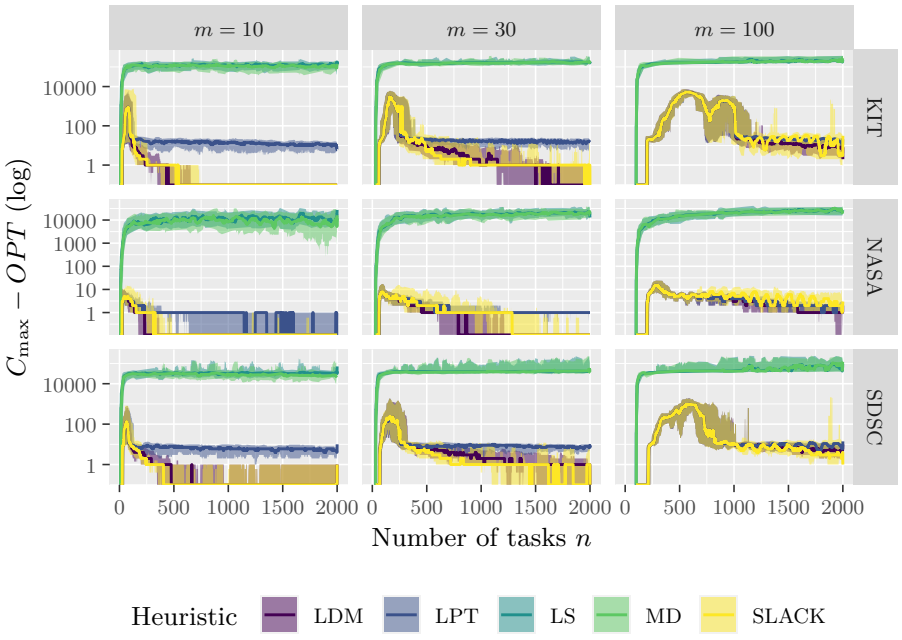


Figure 2.7: Absolute error with costs derived from the KIT ForHLR II and NASA Ames iPSC/860 instances (after optimality transformation). Smoothed lines are obtained by using a rolling median with 45 values (each value is set to the median of the 22 values on the left, the 22 on the right and the current one). The ribbons represent the rolling 0.1- and 0.9-quantiles.

marginally decreases the makespan of LS. We can also see that when n grows, the absolute error of LPT seems to be on par with the one of SLACK. In [**della2020**], SLACK was found to perform generally better than LPT for some synthetic instances. For our realistic instances, it only seems to be true when the number of processors remains small. SLACK and LDM have very similar performances, as their curves often overlap.

2.6 Conclusion

Given various probability distributions, we have evaluated the performance of five heuristics, among which the classical LPT heuristic, the more recent SLACK heuristic, and the LDM heuristic. The literature already contains important theoretical results either in the form of different kinds of stochastic convergence to optimality or with a convergence rate. To the best of our knowledge, this work is the first to empirically assess the tightness of a theoretical convergence rate for LPT, and to study the complexity of LPT with a small number of tasks. Furthermore, we consider a novel definition of uniformity for the cost distribution: for a given total work, any integer composition can be drawn with the same probability, which leads to dependent random costs. This distribution is further enhanced by considering a subset of the decompositions that constrains the minimum cost. We prove the convergence in probability of LPT and four other heuristics with these distributions as well. Finally, we empirically analyze the convergence with realistic distributions obtained through traces. All these results contribute to understand the excellent performance of LPT in practice.

Future work will consist in obtaining stronger convergence theoretical results. For instance, existing results only consider that the number of tasks n tends to infinity. The impact of a varying number of processors m could be explored.

Also, this work is the first attempt to consider dependent cost distributions, but many such distributions exist and could be explored. For instance, the same application consisting of a given set of tasks can be executed with different input size. The tasks could thus often have the same profile to a given multiplying factor. Finally, the novel distribution in this chapter presents a minimum cost. Existing convergence results for independent distributions could probably be extended to consider costs with a similar minimum value. For instance, the worst-case ratio for LPT is achieved with costs $\frac{1}{3}$ and $\frac{1}{2}$. The uniform distribution $\mathcal{U}(\frac{1}{3}, \frac{1}{2})$ could thus present some challenges.

Data availability

The datasets and code generated during and/or analyzed during the study of this chapter are available in the Figshare repository: https://doi.org/10.6084/m9.figshare.19317773.

Chapter 3

List and shelf schedules for independent parallel tasks to minimize the energy consumption with discrete or continuous speeds

Scheduling independent tasks on a parallel platform is a widely-studied problem, in particular when the goal is to minimize the total execution time, or makespan ($P||C_{\max}$ problem in Graham's notations). Also, many applications do not consist of sequential tasks, but rather parallel tasks, either rigid, with a fixed degree of parallelism, or moldable, with a variable degree of parallelism (i.e., for which we can decide at the execution on how many processors they are executed). Furthermore, since the energy consumption of data centers is a growing concern, both from an environmental and economical point of view, minimizing the energy consumption of a schedule is a main challenge to be addressed. One can then decide, for each task, on how many processors it is executed, and at which speed the processors are operated, with the goal to minimize the total energy consumption. We further focus on co-schedules, where tasks are partitioned into shelves, and we prove that the problem of minimizing the energy consumption remains NP-complete when static energy is consumed during the whole duration of the application. We are however able to provide an optimal algorithm for the schedule within one shelf, i.e., for a set of tasks that start at the same time. Several approximation results are derived, both with discrete and continuous speed models, and extensive simulations are performed to show the performance of the proposed algorithms.

3.1 Introduction

We consider the problem of scheduling independent tasks. Even though this problem has already been widely studied, in particular when aiming to minimize the total execution time (or makespan) for sequential tasks, there remain avenues for improvement for variants of the problem. Using the Graham notations [**Graham79**], the typical problem that is studied is $P||C_{\max}$, i.e., the goal is to minimize the makespan when scheduling independent sequential tasks on a set of identical processors. The decision version of this problem in its simplest form is already NP-complete (it is indeed identical to 2-Partition [**GareyJohnson**] when considering two processors). However, several well-known heuristics lead to very good approximation algorithms, as the classical Longest Processing Time (LPT) heuristic, or even some PTAS or FPTAS algorithms [**Hochbaum88**].

The problem becomes more complicated when dealing with parallel tasks. Now, each task i is a parallel task that executes concurrently on p_i processors. The greedy list scheduling algorithm that gives priority to longest jobs is then known to be a 2-approximation when tasks are *rigid* (p_i is given and fixed) [**garey1975bounds**].

In order to ease the scheduling, it can be useful to group tasks by shelves (or batches, packs, levels, etc.), and then the shelves are scheduled one after the other. All the tasks in a same shelf start their execution at the same time, and the next shelf starts only when all tasks of the previous shelf are done. This is typically referred to as *shelf-scheduling* or *co-scheduling*. Of course, one may then waste time, due to idle resources if tasks do not all take the same time. However, such schedules are easy to implement and they also may have some theoretical guarantees. Indeed, the list scheduling that gives priority to longest jobs is known to be a 3-approximation when imposing the use of shelves [**turek1992approximate**] (recall that it is a 2-approximation without this restriction).

Such co-schedules are also very useful for *moldable* tasks, i.e., tasks whose degree of parallelism p_i can be chosen at execution. For such parallel moldable tasks, an easy way to proceed is to execute tasks sequentially, each task using the whole platform. However, it may be more efficient to group tasks by shelves, since the execution profile of a task may lead to less efficiency when using many processors. While the general problem is NP-hard, Aupy et al. [**gopy2016**] propose an optimal polynomial-time algorithm to decide the processor assignment that minimizes the makespan when there are at most two tasks in a shelf.

While most scheduling problems are focusing on makespan minimization, another core problem is the *energy consumption*. In order to optimize this energy consumption, modern processors can run at different speeds, and their power consumption is then the sum of a static part (the cost for a processor to be turned on) and a dynamic part, which is a strictly convex function of the processor speed. Indeed, the execution of a given amount of work costs more power if a processor runs at a higher speed [**10.1109/IPDPS.2006.1639597**]. More precisely, a processor running at speed s dissipates a power of s^3 Watts [**280894**, **pruhsTCS**, **pow3**, **pow3IPDPS**,

pow3ICPP], hence it consumes an energy of $s^3 \times d$ Joules when operated during d units of time. Faster speeds allow for a faster execution, but they also lead to a much higher (supra-linear) power consumption. A more general model states that the power can be in s^α, where $2 \leq \alpha \leq 3$ [**bambagini2016**]. While minimizing the makespan helps reducing the energy consumption, which increases with execution time, to the best of our knowledge, no study has been aiming at minimizing the energy consumption for shelf schedules.

For the static energy consumption, it depends on the time during which processors are powered. We consider two models: in the *independent* model, each processor is independently powered and can be turned off when not computing, hence the static power is paid only while processors are running. However, in the *simultaneous* model, the platform is turned on as long as one processor is running, hence the static power must also be paid for idle processors.

Our main contributions are the following:

- We formalize the problem of scheduling independent moldable tasks to minimize energy consumption (MinE-Mold problem) with various model variants.
- We prove that the problem can be solved in polynomial time when processors are independently powered, while the problem becomes NP-complete with simultaneously powered processors.
- We establish multiple approximation ratios for both classical list scheduling algorithms, and shelf-based schedules, both with a realistic model where speeds can be chosen in a discrete set, and with the general model where speeds can take any positive real value (continuous speeds).
- We provide an optimal dynamic programming algorithm to minimize the energy consumption of a single shelf, both with the discrete and with the continuous model. The goal is to decide on how many processors to execute each task of the shelf, and at which speed to operate the task.
- We perform an empirical study and we show that, for most instances, a single speed can be used for all tasks without increasing the energy consumption. Also, as expected, shelf-based solutions consume more energy, but they are easier to implement and solutions are derived with a much lower complexity. A comparison with solutions using continuous speeds highlights that further energy savings could be achieved by carefully choosing the processor's speeds.

We first discuss related work in Section 3.2. Next, we detail the model (platform, tasks, energy consumption) and schedules, and we introduce the target optimization problems in Section 3.3. The complexity of the problems is established in Section 3.4. Approximation ratios for MinE-Mold are derived in Section 3.5 with discrete speeds and in Section 3.6 with continuous speeds. Optimal algorithms for a single shelf are provided in Section 3.7. Finally, a comprehensive empirical study is proposed in Section 3.8. We conclude and give hints for future research directions in Section 3.9.

3.2 Related work

Although the problem of minimizing the energy consumption of parallel platforms has been extensively studied, few works propose guaranteed scheduling algorithms for moldable tasks. We first cover approximation algorithms for the problem of minimizing the makespan because our approach relies on such results, even though we rather focus on minimizing the energy consumption. We then discuss heuristics proposed for real-time systems, which consider a model slightly different than ours. Finally, we present some examples of moldable task applications.

Makespan minimization. For the rigid case where the number of processors required by each task is fixed, classical *list scheduling algorithms*, denoted by ListBased, are widely used in the literature for makespan minimization. Tasks are ordered in a priority list and are then scheduled by order of priority, as presented by Garey and Graham [**garey1975bounds**]: any time a processor is idle, the list is scanned in order and the first task that can be executed is started. Another way to say it, the principle of the algorithm is as follows: when resources are released, we see if a task can be started right now. If it is the case, we start it. If several tasks can be started, we take the one with the highest priority, given by an ordering of the tasks in a list. ListBased is a 2-approximation for the makespan. More precisely, it is a $2\times\max(\frac{W}{p}, t_{\max})$-approximation, where $\frac{W}{p}$ is the average work and $t_{\max}$ is the execution time of the longest task.

Coffman et al. [**coffman1980**] made a landmark paper proving the approximation ratio of several *shelf-based algorithms* when considering rigid tasks only. They introduce the problem as a two-dimensional packing problem and focus on asymptotic performance bounds for the makespan. They show that the performance bounds of classic bin-packing heuristics Next-Fit Decreasing and First-Fit Decreasing are 3 and 2.7, respectively.

In [**krischnamurti1992**], Krishnamurti et al. study a problem where processors are partitioned, and each task is submitted to one such partition with the objective to minimize the execution time. The number of partitions is bounded, which limits the maximum number of simultaneous tasks.

In [**turek1992approximate**], Turek et al. study the multi-shelves problem, still for makespan minimization. They first propose an allocation strategy for rigid tasks, and then use this strategy on several "allocation candidates" for the moldable case. This first strategy involves *co-schedules*, ShelfBased [**turek1992approximate**], where rigid tasks are partitioned into shelves, and all the tasks in a same shelf begin their execution at the same time (this is equivalent to Next-Fit Decreasing). Tasks are sorted in order of decreasing execution times. Then, tasks are inserted iteratively in the current shelf until the next task cannot be inserted. At this point, a new shelf is created and the process continues. A possible extension consists in allowing backfilling of previous shelves. ShelfBased is a 3-approximation for the makespan. More precisely, it is a $\left(\frac{2W}{p} + t_{\max}\right)$-approximation. To deal with moldable tasks, the authors also present an overall design [**turek1992approximate**] that works as follows. First, a task and a number of processors are selected, and we assume that all tasks will complete before this one. Then, the number of processor is chosen for all other tasks so that their work is

minimized. Finally, this instance is solved as if tasks were rigid (with a fixed number of processors. All pairs of initial task and number of processors are tried. Turek et al. also designed a 2.7-approximation for the multi-shelves problem, with a fixed number of shelves [**turek1992scheduling**]. However, this algorithm is exponential in the number of shelves.

Aupy et al. go beyond the problem of minimizing the makespan, by tackling the problem of optimizing the power consumption, the makespan and the reliability [**aupy2012**]. However, they consider dependent non-parallel tasks, which is a setting completely different from ours, since we focus on parallel tasks. They show that most problems are NP-hard and propose heuristics.

Real-time systems. Finally, several works have been proposed in the context of real-time systems with moldable tasks, power constraints and deadlines. The closest to the problem that we target considers level-based scheduling (similar to shelves) with rigid or moldable tasks [**kong2011**]. They propose heuristics that extend bin-packing ones such as First-Fit Decreasing, Best-Fit Decreasing, etc. Most other works related to real-time systems propose heuristics [**xu2012**, **zahaf2017**, **litzinger2019**]. In this thesis, we do not consider deadlines and we investigate algorithms with guarantees.

Moldable Tasks. Most distributed algorithms have a time complexity that depends on the number of processors and therefore correspond to moldable tasks. This is the case for very classical algorithms such as the Fast Fourier Transform or the product of matrices by Strassen's method. For a more recent example, one can for instance point out distributed algorithms for generating samples from a large tabular [**DBLP:journals/toms/SandersLHSD18**]. In the context of biological applications, E. Saule et al. have shown how to use moldable tasks to tackle the short sequence mapping problem [**DBLP:journals/jpdc/SauleBC12**]. Other applicative examples of moldable tasks on stream algorithms are developed in [**DBLP:journals/vlsisp/KellerLK22**].

Overall, we are not aware of any paper tackling directly the problem that we consider in this chapter, namely the problem of scheduling independent moldable tasks to minimize the energy consumption, under different model variants (shelf-based solutions, discrete and continuous speeds, ...), and hence we were not able to directly compare our proposed approach to any algorithm coming from related work.

3.3 Model

We first describe the platform model (Section 3.3.1), the task model (Section 3.3.2), the energy model (Section 3.3.3), before formally defining general schedules, single-speed schedules and co-schedules in Section 3.3.4. Finally, we introduce the target optimization problems in Section 3.3.5. Table 3.1 summarizes the main notations used throughout the chapter.

Notation	Quantity
p	Number of processors
P_{stat}	Static Power
S	Set of available speeds on the processors
T_i	Task number i
$w_{i,j}$	Total work for the execution of task i on j processors
$t_{i,j,s}$	Execution time of task i on j processors at speed s
$t_{i,j}$	Execution time of task i on j processors at speed $s = 1$
$a_{i,j,s}$	Area of task i on j processors at speed s
λ	A schedule
$p_i(\lambda)$	Number of processors allocated to task i in schedule λ
$s_i(\lambda)$	Speed of the processors allocated to task i in schedule λ
$t_{\max}(\lambda)$	Execution time of the longest task in schedule λ
$C_{\max}(\lambda)$	Makespan of schedule λ
$W(\lambda)$	Sum of the work of all tasks in schedule λ
$T_{dyn}(\lambda)$	Sum of the execution times of all tasks in schedule λ
$A_{dyn}(\lambda)$	Sum of the areas of all tasks in schedule λ
$A_{stat}(\lambda)$	Sum of the times during which processors are powered in schedule λ

Table 3.1: List of notations.

3.3.1 Platform

The target platform consists in p identical processors, whose frequency can be scaled using DVFS (Dynamic Voltage and Frequency Scaling).

These processors have a static power P_{stat} and a set $S = \{s_1, s_2, \ldots, s_k\}$ of possible speeds (or frequencies). For convenience, we let $s_{\min} = s_1$ and $s_{\max} = s_k$ be the minimum and maximum speeds. Indeed, current processors have a set of predefined speeds (or frequencies), which correspond to different voltages that the processor can be subjected to [**Okuma2001**] (*discrete* model).

For the sake of completeness, we also consider the *continuous* model, where processors may be operated at any speed ($S = \mathbb{R}_+^*$). While this model is unrealistic (even though the number of available frequencies tends to be large in modern processors), it is theoretically appealing [**BKP07**]. Also, a study of the problem without a constrained set of speeds allows for a better understanding of how to choose the available speeds during the design of a processor.

In our model, we don't consider the possibility of switching frequencies during the execution of a given task as it would never provide better solutions. The reason for that is that, due to the convexity of the function describing the energy consumption, taking the average speed of a task as its constant speed always incurs a lower energy consumption for the same execution time. Two different tasks, however, can be executed at different frequencies even if they are scheduled on a same processor. This remark would not necessarily remain true if we were to remove the full clairvoyance on the execution times

of the tasks or if we were to add uncertainty to the energy model.

3.3.2 Tasks

We consider a set of n moldable tasks $\{T_1, T_2, \ldots, T_n\}$ with respective execution profiles $(w_{i,j})_{i\in[\![1,n]\!],j\in[\![1,p]\!]}$, where $w_{i,j}$ is the total work required to execute T_i on j processors. The work is the total number of elementary operations to be executed by the processors. If executed at a speed of one, the time per processor is then $t_{i,j} = \frac{w_{i,j}}{j}$.

We assume that:

- $\forall i, (t_{i,j})_j$ is non-increasing in j (the more processors there are, the less time it will take per processor);
- $\forall i, (w_{i,j})_j$ is non-decreasing in j (when using more processors, there is more overhead due to the parallelization, which is a common assumption [**blazewicz2001**, **benoit17**]).

The algorithms presented in this chapter do not require these two usual assumptions, however having them simply allows us to get better time complexities. For example, in the case where some allocations of processors are not possible for some tasks, the corresponding processing time would be infinite and both assumptions could not hold.

Furthermore, for task T_i $(1 \leq i \leq n)$,

- p_i is the number of allocated processors;
- s_i is the speed of the processors during their execution;
- $t_{i,p_i,s_i} = \frac{t_{i,p_i}}{s_i} = \frac{w_{i,p_i}}{s_i \times p_i}$ is the execution time;
- $a_{i,p_i,s_i} = t_{i,p_i,s_i} \times p_i = \frac{w_{i,p_i}}{s_i}$ is the area of the rectangle representing this task.

3.3.3 Energy consumption

The energy consumption consists first of a static part, which corresponds to the power consumed when processors are turned on. The static power is denoted P_{stat}, and the corresponding static energy consumption on each processor is $t_{stat} \times P_{stat}$, where t_{stat} is the duration during which the processor is powered.

There is also a dynamic energy consumption, directly related to the speed s at which the processor operates, and the time t_{dyn} spent computing (which may be equal to or smaller than the time t_{stat}). Using a general model, the dynamic energy consumption is $t_{dyn} \times s^\alpha$ [**bambagini2016**], where $\alpha > 1$ (in general, $2 \leq \alpha \leq 3$). Hence, for task T_i, the dynamic energy consumption on each processor is $t_{i,p_i,s_i} \times s_i^\alpha$ (since $t_{dyn} = t_{i,p_i,s_i}$), and the total dynamic energy consumption for the task is $a_{i,p_i,s_i} \times s_i^\alpha$ (the same energy is consumed by each of the p_i processors operating task T_i).

The case where $t_{stat} = t_{dyn}$ is the *independent* model, where each processor is independently powered, and hence turned off when it is not computing. We also consider the *simultaneous* model, where the whole platform remains powered as long as at least one processor is executing ($t_{stat} = C_{\max}$).

3.3.4 Schedules

Given a computational platform and a set of moldable tasks as described above, a schedule λ is a function that maps each task T_i to a tuple (M_i, s_i, δ_i), where M_i is the set of processors assigned to T_i (hence the number of processors assigned to the task is $p_i = |M_i|$), s_i is the speed of these processors to execute T_i, and $\delta_i \geq 0$ is the starting time of T_i. Moreover, λ must verify the following conditions:

- There exists i such that $\delta_i = 0$ (there is a task starting at time 0);
- If tasks T_i and $T_{i'}$ ($1 \leq i, i' \leq n$ and $i \neq i'$) are such that $M_i \cap M_{i'} \neq 0$, then $[\delta_i, \delta_i + t_{i,p_i,s_i}] \cap [\delta_{i'}, \delta_{i'} + t_{i',p_{i'},s_{i'}}] = \emptyset$ (a processor cannot be used for two different tasks at the same time).

We also have the following aggregated quantities depending on a schedule λ:

- $C_{\max}(\lambda) = \max_i\{\delta_i + t_{i,p_i,s_i}\}$ is the makespan (or total execution time);
- $W(\lambda) = \sum_{i=1}^{n} w_{i,p_i}$ is the cumulative work;
- $T_{dyn}(\lambda) = \sum_{i=1}^{n} t_{i,p_i,s_i}$ is the cumulative execution time of all tasks;
- $A_{dyn}(\lambda) = \sum_{i=1}^{n} a_{i,p_i,s_i}$ is the cumulative execution time on all processors;
- $A_{stat}(\lambda)$ is the cumulative time on all processors during which they are powered. It is either $A_{dyn}(\lambda)$ in the *independent* model, or it is $p \times C_{\max}(\lambda)$ in the *simultaneous* model.

We can then express the total energy consumption of a schedule λ as:

$$E(\lambda) = \sum_{i=1}^{n} a_{i,p_i,s_i} \times s_i^{\alpha} + A_{stat}(\lambda) \times P_{stat}.$$

If there is no ambiguity on λ, we write $C_{\max}$ for $C_{\max}(\lambda)$; and similarly for related quantities (W, T_{dyn}, A_{dyn}, etc.).

Finally, we pay a particular attention to two classes of particular schedules:

- *Single-speed* schedules are schedules such that all the speeds are equal for all tasks, i.e., for $1 \leq i \leq n$, $s_i = s \in S$.
- *Co-schedules* are organized as shelves, as motivated in Section 3.1. A co-schedule consists of a partition of the tasks into shelves and such that:
  - If two tasks are in the same shelf, then they start their execution at the same time;
  - If two tasks are not in the same shelf, then one finishes its execution before the other one starts.

Problem	Processors per task	Speeds	Static area A_{stat}	Constraint
MinE-Mold-Indep	Variable $p_i \in \mathbb{N}$	Finite set S	$A_{stat} = A_{dyn}$	$\emptyset$
MinE-Mold	Variable $p_i \in \mathbb{N}$	Finite set S	$A_{stat} = p \times C_{\max}$	$\emptyset$
MinE-Rig	Fixed $p_i \in \mathbb{N}$	Finite set S	$A_{stat} = p \times C_{\max}$	$\emptyset$
MinE-Mold-Cont	Variable $p_i \in \mathbb{N}$	Interval	$A_{stat} = p \times C_{\max}$	$\emptyset$
MinE-Rig-Cont	Fixed $p_i \in \mathbb{N}$	Interval	$A_{stat} = p \times C_{\max}$	$\emptyset$
MinE-OneShelf	Variable $p_i \in \mathbb{N}$	Finite set S	$A_{stat} = p \times C_{\max}$	$\sum_{i=1}^{n} p_i \leq p$

Table 3.2: List of problems (multiple speeds).

3.3.5 Optimization problems

The general problem is MinE-Mold: Given n moldable tasks, their execution profiles $(w_{i,j})$, p processors and their speeds S, the goal is to find a schedule that minimizes the total energy consumption. We focus mainly on the case with a set of discrete speeds (*discrete* model) and simultaneously-powered processors (*simultaneous* model). Variants with continuous speeds and/or independently-powered processors are referred to by adding Cont or Indep to the problem name (hence, MinE-Mold-Cont-Indep is the problem with both variants, while MinE-Mold-Indep is the problem with discrete speeds and independently-powered processors).

We also consider the variant of the problem with *rigid* tasks, i.e., when the speed s_i and the number of processors per task p_i are fixed (MinE-Rig problem), again by default with discrete speeds and the *simultaneous* model.

Moreover, we add SS to refer to the variant of any problem where the same speed must be selected for each task (see *single-speed* schedules above).

Finally, since we are interested in co-schedules, we consider the more constrained problem with *a single shelf*, i.e., all tasks must start at time 0 and be executed concurrently. The corresponding problem is MinE-OneShelf: Given a set of n tasks and p processors, the goal is to minimize the energy consumption knowing that all tasks start at time 0 ($\delta_i = 0$ for $1 \leq i \leq n$). Solving this particular problem will help us derive efficient co-schedules for the general MinE-Mold problem. More precisely, a solution to MinE-OneShelf is an assignment $((p_i)_{i\in[\![1,n]\!]}, (s_i)_{i\in[\![1,n]\!]})$ such that:

- $\forall i \in [\![1, n]\!]$, task T_i is executed on $p_i \geq 1$ processors at speed $s_i \in S$;
- $\sum_{i=1}^{n} p_i \leq p$ (at most p processors are used, since all tasks execute concurrently).

All problems (with the multiple speed variant) are summarized in Table 3.2.

3.4 Problem complexity

We start with the study of the *independent* model, and we derive that MinE-Mold-Indep can be solved in polynomial time (Section 3.4.1). However, with the more realistic *simultaneous* model, we prove that MinE-Mold is NP-complete (Section 3.4.2).

3.4.1 Optimal algorithm for MinE-Mold-Indep

In the *independent* model, the platform has multiple nodes that are independently powered, which means that each node can individually be turned down at any point of the execution, and not consume any more energy. Therefore, the total energy consumption is the sum of the individual energy consumption of each task. Hence, for each task, we need to decide on how many processors it should be executed, and at which speed, in order to minimize its energy consumption. Recall that we solely focus on energy optimization, and hence we do not have any constraint on the total time to completion.

Since the $(w_{i,j})$'s are non-decreasing in j, we have $a_{i,1,s_i} \leq a_{i,p_i,s_i}$ for all $1 \leq p_i \leq p$. Therefore, a_{i,p_i,s_i} is minimized if a single processor is used (independently of the speed that is chosen). Hence, we set $p_i = 1$, i.e., the task is executed on a single processor (its execution time may be long, but other processors will be turned off and less energy will be consumed, since we have independently-powered processors).

We still need to decide at which speed to execute the task on its single processor. Indeed, there is a tradeoff between executing the task fast to reduce the static energy consumption of the task (the area $a_{i,1,s_i}$, that decreases with s_i), and running at a slower speed to reduce the dynamic energy consumption, which is in s_i^α and hence increases with s_i. The total energy consumption of the task is $a_{i,1,s_i} \times (s_i^\alpha + P_{stat})$, and $a_{i,1,s_i} = \frac{w_{i,1}}{s_i}$. The goal is therefore to find s_i that minimizes $s_i^{\alpha-1} + \frac{P_{stat}}{s_i}$.

The optimal value of s_i (denoted by s_i^{opt}) can then be easily found in $O(|S|)$ if S is a set of discrete speeds, by comparing the values for every possible speed $s_i \in S$. In the continuous case, we remark that the function $f : s \mapsto s_i^{\alpha-1} + \frac{P_{stat}}{s_i}$ is a convex function that reaches its minimum at $s_i^{opt} = \sqrt[\alpha]{\frac{P_{stat}}{\alpha-1}}$, and hence it can be found in $O(1)$.

We can therefore optimize the energy consumption of each task independently, and execute the tasks one after the other. For each task, as shown above, we execute task i on a single processor at speed s_i^{opt}. Again, we focus solely on energy optimization, and the time to completion might be very large in this case (a single processor is used). Of course, one can also schedule the tasks on different processors and achieve the same energy consumption with a smaller total execution time since nodes are independently powered. Anyway, the optimal solution to this problem can therefore be found in polynomial time.

In the rest of this chapter, unless otherwise stated, we focus on simultaneously powered processors (i.e., $A_{stat} = p \times C_{\max}$). It then becomes crucial to use the whole platform and minimize the execution time, since the platform remains powered during the whole execution.

3.4.2 NP-completeness of MinE-Mold

When moving to the *simultaneous* model, it becomes crucial to also minimize the total execution time, since the static energy is consumed during the whole execution. We show that the MinE-Mold problem actually is NP-complete, even when a single speed is available. For the continuous case (MinE-Mold-Cont), the problem is also NP-hard, even though we do not know whether it is in NP or not because of the speeds in $\mathbb{R}_+^*$.

Theorem 3. *The decision problems associated to* MinE-Mold *and* MinE-Mold-SS *are NP-complete, and the decision problems associated to* MinE-Mold-Cont *and* MinE-Mold-Cont-SS *are NP-hard.*

Proof. We first prove that the decision problem associated to MinE-Mold is in NP: a certificate is a schedule, i.e., the number of processors and the speed of each task, as well as the starting time of each task, and it is easy to check in polynomial time whether the bound on energy consumption is achieved. However, in the continuous case, the speeds and starting time of tasks might not be in $\mathbb{Q}$, and hence we do not know whether MinE-Mold-Cont is in NP or not.

To prove the NP-hardness, we do a reduction from the problem of 3-Partition [**GareyJohnson**]: Given $3n$ integers $\{a_1, \ldots, a_{3n}\}$ whose sum is $nB = \sum_{i=1}^{3n} a_i$ and with $\frac{B}{4} < a_i < \frac{B}{2}$ for $1 \leq i \leq 3n$, does there exist a partition of $\{1, \ldots, 3n\}$ into n subsets $S_1, \ldots, S_n$, such that $\sum_{i \in S_j} a_i = B$ for $1 \leq j \leq n$?

Let $\mathcal{I}_1$ be an instance of 3-Partition. We create an instance $\mathcal{I}_2$ of MinE-Mold (or MinE-Mold-Cont) with n processors, and $3n$ tasks that cannot be parallelized, i.e., their execution time is not improved when using more than one processor. Hence, task i ($1 \leq i \leq 3n$) is such that $t_{i,j} = a_i$ for $1 \leq j \leq n$. Furthermore, we have $P_{stat} = 2$ and $\alpha = 3$. In the discrete version of the problem, there is a single speed $S = \{1\}$. Finally, we set the bound on total energy consumption for $\mathcal{I}_2$ to $3nB$.

First, note that it is always better to execute each task on a single processor. Indeed, if a task is executed on more than one processor, it takes the same time to execute but consumes additional energy. Second, in the continuous case, if a single task is considered, it should be executed at speed $s_i^{opt} = \sqrt[\alpha]{\frac{P_{stat}}{\alpha - 1}}$ as shown in Section 3.4.1 for the independent model, which corresponds to a speed of one since $P_{stat} = 2$ and $\alpha = 3$. The use of another speed leads to a higher energy consumption for this task. Hence, assuming that each task is executed at speed 1 (both in the discrete and continuous cases), we obtain a dynamic energy consumption of a_i for task i, and a total dynamic energy consumption of nB. The static energy consumption depends on the total execution time t, and it is $P_{stat} \times t \times n$. If there is no idle time, and hence no waste of static energy, the time $t \times n$ also corresponds to the total time spent executing the tasks as in the independent model, which is nB as for the dynamic energy consumption. We are now ready to prove the equivalence of solutions.

If $\mathcal{I}_1$ has a solution, we execute tasks of a same subset S_j onto processor j, for $1 \leq j \leq n$. Each processor completes in time B, and the static energy consumption is $2nB$, hence a total energy consumption of $3nB$ (static energy plus dynamic energy). Therefore, $\mathcal{I}_2$ has a solution.

If $\mathcal{I}_2$ has a solution, we define S_j as the set of tasks executed on processor j. Since the energy consumption is not greater than $3nB$, each task must be executed at speed 1 on a single processor, otherwise the sum of the energy consumption of each task (as in the independent model) would exceed $3nB$, and lead to a contradiction. Indeed, the energy consumption with the simultaneous model is at least as high as the one with the independent model as it may account for extra static energy consumption due to some

idle time of processors. Given that each task is executed at speed 1, the total dynamic energy consumption is nB and the static energy consumption cannot exceed $2nB$. This means that the total execution time must be such that $t \leq B$, and the sum of the a_i's in each subset S_j cannot exceed B. Therefore, $\mathcal{I}_1$ has a solution, which concludes the proof. □

3.5 Approximation ratios with discrete speeds

To solve MinE-Mold, we extend a strategy [**turek1992approximate**] that transforms a moldable instance into multiple rigid ones by fixing the number of processors (and possibly the speed) of each task. Then, each rigid instance is solved with a heuristic, for example the ones that we mentioned earlier, ListBased (the classical list-based scheduling where no processor is left idle if a task can be started, which is a 2-approximation algorithm for the total execution time, or makespan) or ShelfBased (rigid tasks are partitioned into shelves, and all the tasks in a same shelf begin their execution at the same time, which is a 3-approximation algorithm for makespan).

We thus start by considering the rigid case (MinE-Rig problem) and derive approximation results for energy consumption. Moreover, we first consider a simplified version of the problem where all tasks have the same speed (Section 3.5.1), before moving to the general case (Section 3.5.2). By convention, λ^{OPT} refers to the optimal schedule that minimizes the energy consumption and $\lambda_\bullet$ to the schedule at speed $s_\bullet$ with a guaranteed bound on the energy consumption. Moreover, for a rigid instance, λ^* is the schedule with minimum makespan and $\lambda_{\mathcal{A}}$ a schedule with a guaranteed bound on the makespan.

3.5.1 Processors with a single speed ($s_i = s$)

We first consider the case where the speed must be the same for all tasks, i.e., $s_i = s$ for $1 \leq i \leq n$. The energy simplifies as:

$$E = \sum_{i=1}^{n} a_{i,p_i,s_i} \times s^\alpha + A_{stat} \times P_{stat},$$

with $\alpha > 1$.

Rigid case

We start with the rigid case, which means that, for $1 \leq i \leq n$, the number of processors p_i for task i is fixed. Hence, the workload for task i is also known (w_{i,p_i}). Moreover, for a given schedule λ, all the s_i's are equal to s_λ. The tuple $\lambda(i)$ is hence denoted as $(M_i, s_\lambda, \delta_i)$.

Given any $\rho > 0$, we denote by $\rho\lambda$ the schedule associating to each task i the tuple $(M_i, \rho \times s_\lambda, \frac{\delta_i}{\rho})$, i.e., the speed is scaled by a factor ρ, and the starting times are adjusted accordingly, without any modification in the processor allocation. One can easily check that $\rho\lambda$ is also a rigid single-speed schedule.

Two schedules λ_1 and λ_2 are equivalent, denoted $\lambda_1 \sim \lambda_2$, if there exists $\rho > 0$ such that $\lambda_1 = \rho\lambda_2$. The relation $\sim$ is an equivalence relation. The equivalence class of λ is denoted $[\lambda]$.

Recall that $C_{\max}(\lambda) = \frac{A_{stat}(\lambda)}{p}$ is the makespan: it is the total duration during which the whole system is powered. For convenience, we define the following quantity:

$$K_{[\lambda]} = s_\lambda \times C_{\max}(\lambda).$$

It is easy to see that this is a constant for the equivalence class of λ; indeed, given any $\rho > 0$, $C_{\max}(\rho\lambda) = \frac{C_{\max}(\lambda)}{\rho}$, and $s_{\rho\lambda} = \rho \times s_\lambda$. This is used as makespan that is normalized to the speed.

The goal of this section is to prove the following theorem. The idea consists in considering algorithms with a given approximation ratio on the makespan and show how these ratios extend to the energy minimization. In particular, we consider the same allocation as the one returned by the approximation algorithm but with a speed that minimizes the energy consumption.

Theorem 4. *In the rigid single-speed context (MinE-Rig-SS) and assuming that there exists an algorithm $\mathcal{A}$ that yields a c-approximation of the optimal makespan, we can compute in polynomial time a schedule consuming at most c times the optimal energy.*

Proof. In the rigid case, we have $\sum_{i=1}^{n} a_{i,p_i,s_i} = \sum_{i=1}^{n} \frac{w_{i,p_i}}{s_i}$. The cumulative work $W = \sum_{i=1}^{n} w_{i,p_i}$ is independent of the schedule λ. We can then write the energy consumption of λ as:

$$E(\lambda) = \frac{W}{s_\lambda} \times s_\lambda^\alpha + A_{stat}(\lambda) \times P_{stat} \tag{3.1}$$

$$= W \times s_\lambda^{\alpha-1} + p \times C_{\max}(\lambda) \times P_{stat}. \tag{3.2}$$

Let λ^{OPT} be a single-speed schedule minimizing the energy consumption. Let us denote by $\lambda_{\mathcal{A}}$ the schedule returned by $\mathcal{A}$. Let $s_\bullet \in S$ be a speed for which $\min_{\lambda \in [\lambda_{\mathcal{A}}]} E(\lambda)$ is attained. We analyze the schedule $\lambda_\bullet$ defined by the same allocation as $\mathcal{A}$, but with the speed $s_\bullet$. Its makespan is $C_{\max}(\lambda) = \frac{s_{\lambda_{\mathcal{A}}}}{s_\bullet} \times C_{\max}(\lambda_{\mathcal{A}})$.

Note that if λ^* is a single-speed schedule minimizing the makespan, then $C_{\max}(\lambda_{\mathcal{A}}) \leq c \times C_{\max}(\lambda^*)$ because $\mathcal{A}$ yields a c-approximation of the optimal makespan. Therefore, $K_{[\lambda_{\mathcal{A}}]} \leq c \times K_{[\lambda^*]}$ (necessarily $s_{\lambda_{\mathcal{A}}} \leq s_{\lambda^*} = s_{\max}$). Moreover, for any single-speed schedule λ, $K_{[\lambda^*]} \leq K_{[\lambda]}$, otherwise by running λ at speed $s_{\max}$, we would get a schedule with a lower makespan than λ^*, which would contradicts the fact that λ^* is optimal for the makespan.

$$\begin{aligned}
\min_{\lambda \in [\lambda_{\mathcal{A}}]} E(\lambda_\bullet) &= W \times s_\bullet^{\alpha-1} + \frac{s_{\lambda_{\mathcal{A}}}}{s_\bullet} p \times C_{\max}(\lambda_{\mathcal{A}}) \times P_{stat} && \text{Equation (3.2)} \\
&= W \times s_\bullet^{\alpha-1} + p \times \frac{K_{[\lambda_{\mathcal{A}}]}}{s_\bullet} \times P_{stat} && \text{definition of } K_{[\lambda_{\mathcal{A}}]} \\
&\leq W \times s_{\lambda^{\mathrm{OPT}}}^{\alpha-1} + p \times \frac{K_{[\lambda_{\mathcal{A}}]}}{s_{\lambda^{\mathrm{OPT}}}} \times P_{stat} && \text{optimality of } s^*_{[\lambda_{\mathcal{A}}]}
\end{aligned}$$

$$
\begin{aligned}
&\leq W \times s_{\lambda^{\mathrm{OPT}}}^{\alpha-1} + c \times p \times \frac{K_{[\lambda^*]}}{s_{\lambda^{\mathrm{OPT}}}} \times P_{stat} && K_{[\lambda_{\mathcal{A}}]} \leq c \times K_{[\lambda^*]} \\
&\leq W \times s_{\lambda^{\mathrm{OPT}}}^{\alpha-1} + c \times p \times \frac{K_{[\lambda^{\mathrm{OPT}}]}}{s_{\lambda^{\mathrm{OPT}}}} \times P_{stat} && K_{[\lambda^*]} \leq K_{[\lambda^{\mathrm{OPT}}]} \\
&\leq W \times s_{\lambda^{\mathrm{OPT}}}^{\alpha-1} + c \times p \times C_{\max}(\lambda^{\mathrm{OPT}}) \times P_{stat} && \text{definition of } K_{[\lambda^{\mathrm{OPT}}]} \\
&\leq c \times W s_{\lambda^{\mathrm{OPT}}}^{\alpha-1} + c \times p \times C_{\max}(\lambda^{\mathrm{OPT}}) \times P_{stat} && c \geq 1 \\
&\leq c \times E(\lambda^{\mathrm{OPT}}), && \text{Equation (3.2)}
\end{aligned}
$$

thus proving the theorem. □

Moldable case

The algorithm for MinE-Mold-SS (Algorithm 1) assumes first that a speed of one is used. First, we select both a task $T_{i'}$ and the number of processors $p_{i'}$ for this task. Let $t_{\max}$ be the longest execution time among all tasks (assuming a speed of one). We assume that this time is achieved with this task (i.e., $t_{\max} = t_{i',p_{i'}}$). There are np such selections, and we explore them all. For each value of $t_{\max}$ (i.e., for each pair $(T_{i'}, p_{i'})$), we select the number of processors of each other task to be associated with the lowest work such that $t_{i,p_i} \leq t_{i',p_{i'}}$ still holds. We then solve the rigid instance obtained by fixing the number of processors for each task with ListBased-SS or ShelfBased-SS, and we select the speed that minimizes the energy for the resulting schedule. The final schedule is the one with minimum energy over the np explored possibilities.

Algorithm 1: Algorithm for MinE-Mold-SS

for $(T_{i'}, p') \in \{T_1, \ldots, T_n\} \times \{1, \ldots, p\}$ **do**
  $t_{\max} \leftarrow t_{i',p'}$;
  for $T_i \in \{T_1, \ldots, T_n\}$ **do**
    $p_i \leftarrow \arg\min_{1 \leq j \leq p} j \times t_{i,j}$ such that $t_{i,p_i} \leq t_{\max}$ if it exists;
    $p_i = 0$ otherwise;
  if *all* $p_i \neq 0$ **then**
    Apply a guaranteed algorithm $\mathcal{A}$ on the rigid instance $\{(T_1, p_1), \ldots, (T_n, p_n)\}$ at speed of 1, to get a schedule $\lambda_\bullet^{(T_{i'},p')}$;
    Select the speed $s_\bullet$ that minimizes the energy ;
return the schedule $\lambda_\bullet$ with minimum energy among all the computed $\lambda_\bullet^{(T_{i'},p')}$;

Intuitively, we analyze the approximation ratio of any moldable scheduling algorithm with the following approach based on [**turek1992approximate**]:

- For a given $t_{\max}$, we bound the cumulative work to be executed assuming any task execution duration is bounded by $t_{\max}$.
- We then bound the maximum makespan achievable with a guaranteed algorithm for MinE-Rig-SS.

- Finally, we bound the maximum total energy consumption during this duration.

We can state the main result of this section.

Theorem 5. *We assume that there exists a polynomial-time algorithm $\mathcal{A}$ for* MinE-Rig-SS *that returns a schedule $\lambda_{\mathcal{A}}$ at a speed of one such that $C_{\max}(\lambda_{\mathcal{A}}) \leq a \times \frac{W(\lambda_{\mathcal{A}})}{p} + b \times t_{\max}(\lambda_{\mathcal{A}})$ (resp. $C_{\max}(\lambda_{\mathcal{A}}) \leq \max\left(a \times \frac{W(\lambda_{\mathcal{A}})}{p}, b \times t_{\max}(\lambda_{\mathcal{A}})\right)$). In the moldable single-speed context (*MinE-Mold-SS*), one can compute in polynomial time a schedule $\lambda_\bullet$ that consumes at most $a+b$ (resp. $\max(a,b)$) times the optimal energy.*

Proof. The proof is done for $C_{\max}(\lambda_{\mathcal{A}}) \leq a \times \frac{W(\lambda_{\mathcal{A}})}{p} + b \times t_{\max}(\lambda_{\mathcal{A}})$. The other case (maximum of the two terms instead of sum) is similar.

For any task i and any number p_i of processors, we denote by λ_{i,p_i} the schedule returned by $\mathcal{A}$ on the following rigid instance: for all i', $p_{i'}$ is the integer in $\{1,\dots,p\}$ minimizing $w_{i',p_{i'}}$ under the constraint $t_{i',p_{i'}} \leq t_{i,p_i}$. There are at most np different such schedules and each one can be computed in polynomial time. For each schedule, the selected speed is the one that minimizes the energy consumption. Let $\lambda_\bullet$ be a schedule of $(\lambda_{i,p_i})_{i,p_i}$ (where each task is running at speed s_{i,p_i}) with minimum energy consumption (i.e., the schedule for which $E(\lambda_\bullet) = \min_{i,p_i} E(\lambda_{i,p_i})$).

Let λ^{OPT} be a schedule minimizing the energy (for MinE-Mold-SS). Let i^{OPT} denote the longest task in the optimal schedule λ^{OPT} and $p_{i^{\text{OPT}}}$ denote the number of processors for this longest task (i.e., $t_{\max}(\lambda^{\text{OPT}}) = t_{i^{\text{OPT}},p_{i^{\text{OPT}}}}$). By construction of the λ_{i,p_i}, one has $W(\lambda_g) \leq W(\lambda^{\text{OPT}})$ where $\lambda_g = \lambda_{i^{\text{OPT}},p_{i^{\text{OPT}}}}$. By definition, $t_{\max}(\lambda_g) = t_{\max}(\lambda^{\text{OPT}})$. Thus, at a speed of one, $K_{[\lambda_g]} = C_{\max}(\lambda_g) \leq a \times \frac{W(\lambda_g)}{p} + b \times t_{\max}(\lambda_g) \leq a \times \frac{W(\lambda^{\text{OPT}})}{p} + b \times t_{\max}(\lambda^{\text{OPT}}) \leq (a+b) \times C_{\max}(\lambda^{\text{OPT}}) = (a+b) \times K_{[\lambda^{\text{OPT}}]}$. Finally, remark that a and b are necessarily constants satisfying $a+b \geq 1$ because $\mathcal{A}$ would provide a schedule better than the optimal otherwise.

We have:

$$
\begin{aligned}
E(\lambda_\bullet) &\leq E(\lambda_g) && \text{optimality of } \lambda_\bullet \text{ over all } (\lambda_{i,p_i})_{i,p_i} \\
&\leq E\left(\frac{s_{\lambda^{\text{OPT}}}}{s_{\lambda_g}}\lambda_g\right) && \text{optimality of } s_g \\
&\leq W(\lambda_g) \times s_{\lambda^{\text{OPT}}}^{\alpha-1} + p \times \frac{K_{[\lambda_g]}}{s_{\lambda^{\text{OPT}}}} \times P_{stat} && \text{Equation (3.2)} \\
&\leq W(\lambda^{\text{OPT}}) \times s_{\lambda^{\text{OPT}}}^{\alpha-1} + (a+b) \times p \times \frac{K_{[\lambda^{\text{OPT}}]}}{s_{\lambda^{\text{OPT}}}} \times P_{stat} && \text{approximation ratio of } \mathcal{A} \\
&\leq (a+b) \times W(\lambda^{\text{OPT}}) \times s_{\lambda^{\text{OPT}}}^{\alpha-1} + (a+b) \times p \times \frac{K_{[\lambda^{\text{OPT}}]}}{s_{\lambda^{\text{OPT}}}} \times P_{stat} && a+b \geq 1 \\
&\leq (a+b) \times E(\lambda^{\text{OPT}}) && \text{Equation (3.2)},
\end{aligned}
$$

which concludes the proof. □

It has already been proved that ListBased-SS is an algorithm that outputs a schedule λ at a speed of one such that $C_{\max}(\lambda) \leq \max\left(2 \times \frac{W(\lambda)}{p}, 2 \times t_{\max}(\lambda)\right)$ [**garey1975bounds**], so by applying Theorem 5 with a maximum and $a = 2$ and $b = 2$, we show that ListBased-SS is a 2-approximation algorithm for the energy.

As for ShelfBased-SS, it is an algorithm that outputs a schedule λ such that $C_{\max}(\lambda) \leq 2 \times \frac{W(\lambda)}{p} + t_{\max}(\lambda)$ [**turek1992approximate**], so by applying Theorem 5 with a sum and $a = 2$ and $b = 1$ [**turek1992approximate**], we show that ShelfBased-SS is thus a 3-approximation algorithm for the energy.

3.5.2 Processors with different speeds for each task

When generalizing to multiple speeds, the approach is close to the one used for the single-speed problem where all tasks are executed at the same speed (see Algorithm 1); the corresponding algorithm is detailed in Algorithm 2. Note that in the case where the speeds of the tasks are already determined, the dynamic area A_{dyn} is equivalent to the work from [**turek1992scheduling**], which was denoted by W in Theorem 5.

Algorithm 2: Algorithm for MinE-Mold, with multiple speeds

for $(T_{i'}, p', s') \in \{T_1, \ldots, T_n\} \times \{1, \ldots, p\} \times \{s_1, \ldots, s_k\}$ **do**
  $t_{\max} \leftarrow t_{i',p',s'}$;
  for $T_i \in \{T_1, \ldots, T_n\}$ **do**
    $p_i, s_i \leftarrow \arg\min_{1 \leq j \leq p, s \in S} a_{i,j,s} \times s^\alpha + a_{i,j,s} \times P_{stat}$ such that $t_{i,p_i,s_i} \leq t_{\max}$ if it exists;
    $p_i, s_i \leftarrow 0, 0$ otherwise.
  if *all* $p_i, s_i \neq 0, 0$ **then**
    Apply a guaranteed algorithm $\mathcal{A}$ on the rigid instance $\{(T_1, p_1, s_1), \ldots, (T_n, p_n, s_n)\}$ at speed of 1, to get a schedule $\lambda_\bullet^{(T_{i'}, p', s')}$;
return the schedule $\lambda_\bullet$ with minimum energy among all the computed $\lambda_\bullet^{(T_{i'}, p_{i'}, s_{i'})}$;

Theorem 6. *We assume that there exists a polynomial-time algorithm $\mathcal{A}$ for MinE-Rig-SS that returns a schedule $\lambda_\mathcal{A}$ such that $C_{\max}(\lambda_\mathcal{A}) \leq a \times \frac{A_{dyn}(\lambda_\mathcal{A})}{p} + b \times t_{\max}(\lambda_\mathcal{A})$ (resp. $C_{\max}(\lambda_\mathcal{A}) \leq \max\left(a \times \frac{A_{dyn}(\lambda_\mathcal{A})}{p}, b \times t_{\max}(\lambda_\mathcal{A})\right)$) with $1 \leq a$. In the general moldable context (MinE-Mold), one can compute in polynomial time a schedule $\lambda_\bullet$ that consumes at most $a + b$ (resp. $\max(a, b + 1)$) times the optimal energy.*

Proof. We consider in this proof the max case for $\mathcal{A}$. The proof is similar for the sum.

For any task i, any number p_i of processors and any speed s_i, we consider the following rigid instance: for all i', we select $(p_{i'}, s_{i'}) \in \{1, \ldots, p\} \times S$ that minimizes the energy consumption of task i, $a_{i',p_{i'},s_{i'}} \times s_{i'}^\alpha + a_{i',p_{i'},s_{i'}} \times P_{stat}$ where $a_{i',p_{i'},s_{i'}}$ is the area of the rectangle representing task i, under the constraint $t_{i',p_{i'},s_{i'}} \leq t_{i,p',s}$. The schedule

returned by $\mathcal{A}$ for this problem is denoted λ_{i,p',s_i}. There are at most $np|S|$ different such schedules and each one can be computed in polynomial time. Let $\lambda_\bullet$ be a schedule among the $(\lambda_{i,p_i,s_i})_{i,p_i,s_i}$ minimizing the energy.

Let λ^{OPT} be a schedule minimizing the energy (for MinE-Mold). Let i^{OPT}, $p_{i^{\mathrm{OPT}}}$ and $s_{i^{\mathrm{OPT}}}$ satisfy $t_{\max}(\lambda^{\mathrm{OPT}}) = t_{i^{\mathrm{OPT}},p_{i^{\mathrm{OPT}}},s_{i^{\mathrm{OPT}}}}$. For a schedule λ, set $E_i(\lambda) = a_{i,p_i,s_i}s_i^\alpha + a_{i,p_i,s_i}P_{stat}$. By construction, one has:

$$\sum_i E_i(\lambda_{i^{\mathrm{OPT}},p_{i^{\mathrm{OPT}}},s_{i^{\mathrm{OPT}}}}) \leq \sum_i E_i(\lambda^{\mathrm{OPT}}). \tag{3.3}$$

To simplify the notation, we denote $\lambda_{i^{\mathrm{OPT}},p_{i^{\mathrm{OPT}}},s_{i^{\mathrm{OPT}}}}$ by λ_g. Now,

$$\begin{aligned}
E(\lambda_\bullet) &\leq E(\lambda_{i^{\mathrm{OPT}},p_{i^{\mathrm{OPT}}},s_{i^{\mathrm{OPT}}}}) = E(\lambda_g) \\
&\leq \sum_i a_{i,p_i,s_i}(\lambda_g) \times s_i^\alpha + A_{stat}(\lambda_g) \times P_{stat} && \text{definition of the energy} \\
&\leq \sum_i a_{i,p_i,s_i}(\lambda_g) \times s_i^\alpha + p \times C_{\max}(\lambda_g) \times P_{stat} && \text{definition of } A_{stat} \\
&\leq \sum_i a_{i,p_i,s_i}(\lambda_g) \times s_i^\alpha + p \times \max\left(a \times \frac{A_{dyn}(\lambda_g)}{p}, b \times t_{\max}(\lambda_g)\right) \times P_{stat} && \text{approximation ratio of } \mathcal{A}
\end{aligned}$$

Now, by distributivity, we have one of the two following possibilities:

$$E(\lambda_\bullet) \leq \sum_i a_{i,p_i,s_i}(\lambda_g) \times s_i^\alpha + a \times A_{dyn}(\lambda_g) \times P_{stat} \qquad \text{left-hand side of the max}$$

or

$$E(\lambda_\bullet) \leq \sum_i a_{i,p_i,s_i}(\lambda_g) \times s_i^\alpha + b \times p \times t_{\max}(\lambda_g) \times P_{stat} \qquad \text{right-hand side of the max}$$

We start with the left-hand side of the max:

$$\begin{aligned}
\mathrm{LHS} &\triangleq \sum_i a_{i,p_i,s_i}(\lambda_g) \times s_i^\alpha + a \times A_{dyn}(\lambda_g) \times P_{stat} \\
&\leq a \times (\sum_i a_{i,p_i,s_i}(\lambda_g) \times s_i^\alpha + A_{dyn}(\lambda_g) \times P_{stat}) && 1 \leq a \\
&\leq a \times \sum_i (a_{i,p_i,s_i}(\lambda_g) \times s_i^\alpha + a_{i,p_i,s_i}(\lambda_g) \times P_{stat}) && \text{definition of } A_{dyn} \\
&\leq a \times \sum_i E_i(\lambda_g) && \text{definition of } E_i \\
&\leq a \times \sum_i E_i(\lambda^{\mathrm{OPT}}) && \text{Equation (3.3)} \\
&\leq a \times E(\lambda^{\mathrm{OPT}}) && \sum E_i \leq E
\end{aligned}$$

Now, the right-hand side of the max:

$$\mathrm{RHS} \triangleq \sum_i a_{i,p_i,s_i}(\lambda_g) \times s_i^\alpha + b \times p \times t_{\max}(\lambda_g) \times P_{stat}$$

$$
\begin{aligned}
&\leq \sum_i a_{i,p_i,s_i}(\lambda_g) \times s_i^\alpha + b \times p \times t_{\max}(\lambda^{\mathrm{OPT}}) \times P_{stat} && t_{\max}(\lambda_g) = t_{\max}(\lambda^{\mathrm{OPT}}) \\
&\leq \sum_i a_{i,p_i,s_i}(\lambda_g) \times s_i^\alpha + b \times A_{stat}(\lambda^{\mathrm{OPT}}) \times P_{stat} && p \times t_{\max} \leq A_{stat} \text{ for any given schedule} \\
&\leq \sum_i E_i(\lambda_g) + b \times A_{stat}(\lambda^{\mathrm{OPT}}) \times P_{stat} && a_{i,p_i,s_i}(\lambda_g) \times s_i^\alpha \leq E_i(\lambda_g) \text{ from the definition of } E_i \\
&\leq \sum_i E_i(\lambda_g) + b \times E(\lambda^{\mathrm{OPT}}) && A_{stat} \times P_{stat} \leq E \text{ for any given schedule} \\
&\leq \sum_i E_i(\lambda^{\mathrm{OPT}}) + b \times E(\lambda^{\mathrm{OPT}}) && \text{Equation (3.3)} \\
&\leq E(\lambda^{\mathrm{OPT}}) + b \times E(\lambda^{\mathrm{OPT}}) && \sum E_i \leq E \\
&\leq (b+1) \times E(\lambda^{\mathrm{OPT}})
\end{aligned}
$$

We finally reunite the two sides of the max:

$$
\begin{aligned}
E(\lambda_\bullet) &\leq \max\left(a \times E(\lambda^{\mathrm{OPT}}), (b+1) \times E(\lambda^{\mathrm{OPT}})\right) \\
&\leq \max(a, b+1) \times E(\lambda^{\mathrm{OPT}}),
\end{aligned}
$$

which concludes the proof. □

In this case, the bound is 3 for both ListBased and ShelfBased algorithms.

3.6 Approximation ratios with continuous speeds

We propose a theoretical variation of the problem, where instead of choosing the speed in a finite set of speeds S, we can choose any speed in $\mathbb{R}_+^*$. We call this continuous problem MinE-Mold-Cont and MinE-Rig-Cont, depending on the nature of the tasks. The motivation for this variation is that we can get stronger approximation results through the introduction of continuous speeds. Through this, we hope to get a better understanding of what makes a good processor speed, thus allowing us to better choose the set of speeds S at the creation of a processor.

3.6.1 Rigid case

Similarly to the discrete case, we start by restricting to rigid tasks (MinE-Rig-Cont-SS problem).

Theorem 7. *In the rigid single-speed context with continuous speeds (MinE-Rig-Cont-SS) and assuming that there exists an algorithm $\mathcal{A}$ that yields a c-approximation of the optimal makespan, we can compute in polynomial time a schedule consuming at most $c^{1-\frac{1}{\alpha}}$ times the optimal energy.*

Proof. In the rigid case, we have $\sum_{i=1}^{n} a_{i,p_i,s_i} = \sum_{i=1}^{n} \frac{w_{i,p_i}}{s_i}$. The cumulative work $W = \sum_{i=1}^{n} w_{i,p_i}$ is independent of the schedule λ. With a single speed, we can hence write the energy as:

$$E(\lambda) = W \times s_\lambda^{\alpha-1} + A_{stat}(\lambda) \times P_{stat} \tag{3.4}$$
$$= W \times s_\lambda^{\alpha-1} + p \times C_{\max}(\lambda) \times P_{stat}. \tag{3.5}$$

The problem can be split as two decisions to take:

- The choice of speed s;
- The actual scheduling, i.e., the choice of the time at which we start each task.

To show that these decisions can be taken one after the other, we start with a preliminary lemma comparing the energy consumption of two schedules using the same speed.

Lemma 6. *Let λ_1, λ_2 be two single-speed schedules such that $s_{\lambda_1} = s_{\lambda_2}$. If $E(\lambda_1) \leq E(\lambda_2)$, then for any $\rho > 0$, $E(\rho\lambda_1) \leq E(\rho\lambda_2)$.*

Proof. Using Equation (3.2),

$$E(\lambda_2) - E(\lambda_1) = p \times P_{stat} \times (C_{\max}(\lambda_2) - C_{\max}(\lambda_1)).$$

Furthermore, $C_{\max}(\lambda_1) = \rho \times C_{\max}(\rho\lambda_1)$ and $C_{\max}(\lambda_2) = \rho \times C_{\max}(\rho\lambda_2)$. It follows that:

$$\begin{aligned} E(\lambda_2) - E(\lambda_1) &= p \times P_{stat} \times \rho \times (C_{\max}(\rho\lambda_2) - C_{\max}(\rho\lambda_1)) \\ &= \rho \times (E(\rho\lambda_2) - E(\rho\lambda_1)), \end{aligned}$$

hence proving the lemma. □

Lemma 6 shows that the actual scheduling can be expressed as a two-steps minimization problem: find a schedule λ_0 minimizing the makespan for a given speed. Next, find among $[\lambda_0]$ (using the notations defined in Section 3.5.1) a schedule (i.e., a speed) minimizing the energy consumption.

For a given schedule λ_0, we can compute the optimal speed s as the one that minimizes

$$f(s) = W \times s^{\alpha-1} + p \times \frac{K_{[\lambda_0]}}{s} \times P_{stat},$$

with $K_{[\lambda]} = s_\lambda \times C_{\max}(\lambda)$ as defined in Section 3.5.1. This optimal value of s is

$$s_{[\lambda_0]}^{\text{OPT}} \triangleq \sqrt[\alpha]{\frac{p \times K_{[\lambda_0]}}{(\alpha-1) \times W} \times P_{stat}}.$$

Then, we can write

$$\min_{\lambda \in [\lambda_0]} E(\lambda) = C \times \sqrt[\alpha]{K_{[\lambda_0]}^{\alpha-1} \times W},$$

where $C = \sqrt[\alpha]{(p \times P_{stat})^{\alpha-1}} \times (\sqrt[\alpha]{\alpha-1} + \sqrt[\alpha]{(\alpha-1)^{\alpha-1}})$ is a constant independent of λ_0.

Consequently, we have

$$\begin{aligned} E(\lambda^{\mathrm{OPT}}) &= \min_{[\lambda_0]\ \sim\text{class}} \min_{\lambda \in [\lambda_0]} E(\lambda) \\ &= \min_{[\lambda_0]\ \sim\text{class}} C \times \sqrt[\alpha]{K_{[\lambda_0]}^{\alpha-1} \times W} \\ &= C \times \sqrt[\alpha]{K_{[\lambda^*]}^{\alpha-1} \times W}. \end{aligned}$$

Now, let $\mathcal{A}$ be an algorithm that yields a c-approximation for the makespan. For a given speed, the quantity $K_{[\lambda_{\mathcal{A}}]}$ is proportional to the makespan, so this algorithm outputs a schedule $\lambda_{\mathcal{A}}$ such that

$$K_{[\lambda_{\mathcal{A}}]} \leq c \times K_{[\lambda^*]}.$$

By running this schedule with speed

$$s_{[\lambda_{\mathcal{A}}]}^{\mathrm{OPT}} = \sqrt[\alpha]{\frac{p \times K_{[\lambda_0]}}{(\alpha-1) \times W} \times P_{stat}},$$

we have a schedule such that

$$\begin{aligned} E_{\lambda_{\mathcal{A}}} &= C \times \sqrt[\alpha]{K_{[\lambda_{\mathcal{A}}]}^{\alpha-1} \times W} \\ &\leq C \times \sqrt[\alpha]{\left(c \times K_{[\lambda^*]}\right)^{\alpha-1} \times W} \\ &\leq c^{\frac{\alpha-1}{\alpha}} \times E(\lambda^{\mathrm{OPT}}). \end{aligned}$$

This proves the theorem: an algorithm $\mathcal{A}$, that yields a c-approximation for the makespan and that returns a schedule $\lambda_{\mathcal{A}}$ will yield a $c^{\frac{\alpha-1}{\alpha}}$-approximation for the energy. □

3.6.2 Moldable case

The overall design of the algorithm for MinE-Mold-Cont-SS is similar to the one of the discrete case, see Algorithm 3.

We can state the main result of this section.

Theorem 8. *We assume that there exists a polynomial-time algorithm $\mathcal{A}$ for MinE-Rig-Cont-SS that returns a schedule $\lambda_{\mathcal{A}}$ at a speed of one such that $C_{\max}(\lambda_{\mathcal{A}}) \leq a \times \frac{W(\lambda_{\mathcal{A}})}{p} + b \times t_{\max}(\lambda_{\mathcal{A}})$ (resp. $C_{\max}(\lambda_{\mathcal{A}}) \leq \max\left(a \times \frac{W(\lambda_{\mathcal{A}})}{p}, b \times t_{\max}(\lambda_{\mathcal{A}})\right)$). In the moldable single-speed context with continuous speeds (MinE-Mold-Cont-SS), one can compute in polynomial time a schedule $\lambda_\bullet$ that consumes at most $(a+b)^{1-\frac{1}{\alpha}}$ (resp. $\max(a,b)^{1-\frac{1}{\alpha}}$) times the optimal energy, when running at speed $s_\bullet \stackrel{\Delta}{=} \sqrt[\alpha]{\frac{p \times K_{[\lambda_{\mathcal{A}}]}}{(\alpha-1) \times W} \times P_{stat}}$.*

Algorithm 3: Algorithm for MinE-Mold-Cont-SS

for $(T_{i'}, p') \in \{T_1, \ldots, T_n\} \times \{1, \ldots, p\}$ **do**
  $t_{\max} \leftarrow t_{i',p'}$;
  for $T_i \in \{T_1, \ldots, T_n\}$ **do**
    $p_i \leftarrow \arg\min_{1 \le j \le p} j \times t_{i,j}$ such that $t_{i,p_i} \le t_{\max}$ if it exists;
    $p_i = 0$ otherwise;
  if *all* $p_i \neq 0$ **then**
    Apply a guaranteed algorithm $\mathcal{A}$ on the rigid instance $\{(T_1, p_1), \ldots, (T_n, p_n)\}$ at speed of 1, to get a schedule $\lambda_\bullet^{(T_{i'},p')}$;
    Select the speed $s_\bullet \triangleq \sqrt[\alpha]{\frac{p \times K_{[\lambda_\bullet]}}{(\alpha-1)\times W} \times P_{stat}}$ for this schedule ;
return the schedule $\lambda_\bullet$ with minimum energy among all computed $\lambda_\bullet^{(T_{i'},p')}$;

Proof. We adapt the proof of Theorem 5 to continuous speeds for the case $C_{\max}(\lambda_{\mathcal{A}}) \le a \times \frac{W(\lambda_{\mathcal{A}})}{p} + b \times t_{\max}(\lambda_{\mathcal{A}})$. The other case (maximum of the two terms instead of sum) is similar.

For any task i and any number p_i of processors, we denote by λ_{i,p_i} the schedule returned by $\mathcal{A}$ on the following rigid instance: for all i', $p_{i'}$ is the integer in $\{1, \ldots, p\}$ minimizing $w_{i',p_{i'}}$ under the constraint $t_{i',p_{i'}} \le t_{i,p_i}$. There are at most np different such schedules and each one can be computed in polynomial time. For each schedule, the selected speed is the one that minimizes the energy consumption. Let $\lambda_\bullet$ be a schedule of $(\lambda_{i,p_i})_{i,p_i}$ (where each task is running at speed $s_{i,p_i} = \sqrt[\alpha]{\frac{K_{[\lambda_{i,p_i}]}}{(\alpha-1)\times W} \times P_{stat}}$) with minimum energy consumption.

Let λ^{OPT} be a schedule minimizing the energy (for MinE-Mold-Cont-SS). Let i^{OPT} and $p_{i^{\mathrm{OPT}}}$ denote the task and the number of processors, such that $t_{\max}(\lambda^{\mathrm{OPT}}) = t_{i^{\mathrm{OPT}},p_{i^{\mathrm{OPT}}}}$ in the schedule λ^{OPT} that minimizes the energy consumption. By construction of the λ_{i,p_i}, one has $W(\lambda_g) \le W(\lambda^{\mathrm{OPT}})$ where $\lambda_g = \lambda_{i^{\mathrm{OPT}},p_{i^{\mathrm{OPT}}}}$. As in the rigid context (proof of Theorem 7), we can express the energy for the respective schedules as $E(\lambda_g = C \sqrt[\alpha]{K_{[\lambda_g]}^{\alpha-1} \times W(\lambda_g)}$ and $E(\lambda^{\mathrm{OPT}}) = C \sqrt[\alpha]{K_{[\lambda^{\mathrm{OPT}}]}^{\alpha-1} \times W(\lambda^{\mathrm{OPT}})}$ where $C = \sqrt[\alpha]{(p \times P_{stat})^{\alpha-1}} \times (\sqrt[\alpha]{\alpha - 1} + \sqrt[\alpha]{(\alpha-1)^{\alpha-1}})$.

Consequently $\frac{E(\lambda_\bullet)}{E(\lambda^{\mathrm{OPT}})} \le \frac{E(\lambda_g)}{E(\lambda^{\mathrm{OPT}})} = \sqrt[\alpha]{\frac{K_{[\lambda_g]}^{\alpha-1}}{K_{[\lambda^{\mathrm{OPT}}]}^{\alpha-1}}}$. Now, we have

$$\begin{aligned} K_{[\lambda_g]} &\le a \times \frac{W(\lambda_g)}{p} + b \times t_{\max}(\lambda_g) \\ &\le a \times \frac{W(\lambda^{\mathrm{OPT}})}{p} + b \times t_{\max}(\lambda^{\mathrm{OPT}}) \\ &\le (a+b) \times K_{[\lambda^{\mathrm{OPT}}]} \end{aligned}$$

From that, we finally get

$$\frac{E(\lambda_\bullet)}{E(\lambda^{\mathrm{OPT}})} \leq \sqrt[\alpha]{(a+b)^{\alpha-1}},$$

which concludes the proof. □

Recall that ListBased-SS is an algorithm with a maximum and $a = 2$ and $b = 2$ [**garey1975bounds**], hence it is a $2^{1-\frac{1}{\alpha}}$-approximation algorithm. ShelfBased-SS, an algorithm with a sum and $a = 2$ and $b = 1$ [**turek1992approximate**], is thus a $3^{1-\frac{1}{\alpha}}$-approximation algorithm. For $\alpha = 3$, these approximation ratios become respectively $\sqrt[3]{4} \approx 1.59$ and $\sqrt[3]{9} \approx 2.08$.

3.7 Optimizing for a single shelf

We propose to further optimize co-schedules by designing a polynomial-time algorithm for the MinE-OneShelf problem, i.e., to optimize the execution of a single shelf, both with discrete and continuous speeds. Formally, given a set of n tasks and p processors, the goal is to find an assignment $((p_i),(s_i))_{1\leq i\leq n}$ that minimizes $E = \sum_{i=1}^{n} \left(p_i \times t_{i,dyn} \times s_i^\alpha + p_i \times t_{i,stat} \times P_{stat}\right)$, where

- $t_{i,dyn} = t_{i,p_i,s_i} = \frac{t_{i,p_i}}{s_i}$, and
- $t_{i,stat} = \max_{1\leq i\leq n} t_{i,dyn}$ (*simultaneous* model).

3.7.1 Preliminaries

Since the static energy spent depends on the total length of the shelf (i.e., $\max_{1\leq i\leq n} t_{i,p_i,s_i}$), the algorithm proceeds by fixing the shelf length to $C_{\max}$, and aims at finding the optimal number of processors and speed for each task, such that the time bound $C_{\max}$ is respected and the total energy consumption is minimized.

Hence, for a single processor, given an amount of work w to complete and a length of shelf of $C_{\max}$, we consider the function $OptS(w, C_{\max})$ that returns the optimal speed such that $\frac{w}{s} \leq C_{\max}$ and the energy consumption $w \times s^{\alpha-1} + C_{\max} \times P_{stat}$ is minimized. Since the energy consumption is an increasing function of s for $s \geq 0$, the optimal speed is the smallest speed such that the shelf length is not exceeded. Therefore, $OptS(w, C_{\max}) = \max(s_{\min}, \frac{w}{C_{\max}})$ in the continuous case, and $OptS(w, C_{\max}) = \min\left\{s \in S \mid s \geq \frac{w}{C_{\max}}\right\}$ in the discrete case. In the case no such speed exists (this may happen with discrete speeds), the function returns None. Note that this function can be computed in $\Theta(1)$ in the continuous case, and in $\Theta(\log(|S|))$ in the discrete case by doing a binary search within values of S.

3.7.2 Optimal algorithm for MinE-OneShelf (discrete speeds)

We first focus on the discrete case, i.e., S is the set of possible speeds. The idea is to try every possible duration of the shelf: all possible durations are recorded in the set $\mathcal{T}$,

and then for a given duration $C_{\max} \in \mathcal{T}$, we compute the solution for each set of tasks $T_1, \ldots, T_i$, $i \in [\![1, n]\!]$ and each number of processors $q \in [\![1, p]\!]$.

Let $e_{i,q}$ be the minimum energy consumed by task T_i on q processors, while not exceeding time $C_{\max}$. It is computed by using the function $OptS(t_{i,q}, C_{\max})$, since $t_{i,q}$ is the amount of work on one processor if task T_i is executed on q processors.

We then proceed with a dynamic programming algorithm, to compute $E_{i,q}$, the minimum energy consumption for the first i tasks, when using a total of q processors. The goal is to compute $E_{n,p}$ (using all tasks and all processors). $E_{i,q}$ is recursively defined for $1 \leq i \leq n$ and $1 \leq q \leq p$ as:

$$E_{i,q} = \min_{1 \leq k \leq q-i+1} E_{i-1,q-k} + e_{i,k},$$

with $E_{0,q} = 0$. If there are no tasks left, the energy consumption is null; otherwise we try all possible numbers of processors k for task i, while keeping at least one processor for each of the remaining tasks.

We then take the best possible solution amongst the different possible durations in the set $\mathcal{T}$, and Algorithm 4 provides the corresponding pseudo-code of this dynamic programming algorithm.

Theorem 9. *MinE-OneShelf can be solved optimally in polynomial time (discrete model).*

Proof. Let us prove by induction over $i \in [\![0, n]\!]$ that for all $q \in [\![1, p]\!]$, $E_{i,q}$ is the minimum energy consumed to process the i first tasks with q processors.

Base case For all $q \in [\![0, p]\!]$, the energy consumed to handle no task on q processors is 0, meaning that the $E_{0,q}$ values for $q \in [\![0, p]\!]$ are correct.

Inductive step Let $i \in [\![0, n-1]\!]$, and we assume that $\forall q \in [\![1, p]\!]$, $E_{i,q}$ is correct. The expression of $E_{i+1,q}$ is $\min_{1 \leq k \leq q-i} E_{i,q-k} + e_{i+1,k}$. If task T_{i+1} is given k processors, then the i first tasks will be handled by $q-k$ processors. As the task $i+1$ must be given a number of processors $k \in [\![1, q-i]\!]$, the expression gives the correct value for $E_{i+1,q}$.

It means that $E_{n,p}$ is correct, and therefore that the algorithm is also correct.

The number of total durations is at most $np|S|$, because we must choose a task, the number of processors allocated for this task, and its speed. The complexity of the algorithm is thus $O(n^2p^3|S|)$. □

3.7.3 Optimal algorithm for MinE-OneShelf-Cont (continuous speeds)

We now discuss the case of continuous speeds, hence $S = \mathbb{R}_+^*$. Similarly to the discrete case, the idea is to fix the shelf duration and to solve the problem knowing that the processors will be powered for the duration $C_{\max}$. Because of continuous speeds, we cannot anymore explore all possible times $C_{\max}$, so we fix the duration to $C_{\max} = 1$, and then prove that the optimal assignment is in fact the same for any $C_{\max}$, and the

Algorithm 4: Optimal algorithm for MinE-OneShelf (discrete model)

$\mathcal{T} \leftarrow \emptyset$;
for $i \leftarrow 1$ **to** n **do**
  for $q \leftarrow 1$ **to** p **do**
    for $s \in S$ **do**
      $\mathcal{T} \leftarrow \mathcal{T} \cup \{\frac{t_{i,q}}{s}\}$;
$res \leftarrow \infty$;
for $C_{\max} \in \mathcal{T}$ **do**
  for $i \leftarrow 1$ **to** n **do**
    for $q \leftarrow 1$ **to** p **do**
      $s_{i,q} \leftarrow OptS(t_{i,q}, C_{\max})$;
      if $s_{i,q} =$ None **then**
        $e_{i,q} \leftarrow \infty$;
      else
        $e_{i,q} \leftarrow q t_{i,q} s_{i,q}^{\alpha-1} + q C_{\max} P_{stat}$;
  for $q \leftarrow 0$ **to** p **do**
    $E_{0,q} \leftarrow 0$;
  for $i \leftarrow 1$ **to** n **do**
    for $q \leftarrow i$ **to** p **do**
      for $k \leftarrow 1$ **to** $q - i + 1$ **do**
        if $E_{i-1,q-i} + e_{i,k} < E_{i,q}$ **then**
          $E_{i,q} \leftarrow E_{i-1,q-k} + e_{i,k}$;
  if $E_{n,p} < res$ **then**
    $res \leftarrow E_{n,p}$
return res ;

Algorithm 5: Optimal algorithm for MinE-OneShelf-Cont

for $i \leftarrow 1$ **to** n **do**
  for $q \leftarrow 1$ **to** p **do**
    $s_{i,q} \leftarrow OptS(t_{i,q}, 1)$;
    if $s_{i,q} = \text{None}$ **then**
      $e_{i,q} \leftarrow \infty$;
    else
      $e_{i,q} \leftarrow q t_{i,q} s_{i,q}^{\alpha-1} + q P_{stat}$;
for $q \leftarrow 0$ **to** p **do**
  $E_{0,q} \leftarrow 0$;
for $i \leftarrow 1$ **to** n **do**
  for $q \leftarrow i$ **to** p **do**
    for $k \leftarrow 1$ **to** $q - i + 1$ **do**
      if $E_{i-1,q-k} + e_{i,k} < E_{i,q}$ **then**
        $E_{i,q} \leftarrow E_{i-1,q-k} + e_{i,k}$;
$C_{\max} \leftarrow \sqrt[\alpha-2]{(\alpha-1)(\frac{E_{n,p}}{pP_{stat}} - 1)}$;
return $\frac{E_{n,p} - pP_{stat}}{C_{\max}^{\alpha-1}} + pC_{\max}P_{stat}$;

minimum energy is a function of $C_{\max}$. We finally take the value of $C_{\max}$ that minimizes the energy consumption, see Algorithm 5 and Theorem 10 for the proof of optimality.

Lemma 7. *Let $E_{dyn}(C_{\max})$ be the minimum possible dynamic energy consumption, with the condition that each task must end before $C_{\max}$. Then, for any $C_{\max} \in \mathbb{R}_+^*$, we have:*

$$E_{dyn}(C_{\max}) = \frac{E_{dyn}(1)}{C_{\max}^{\alpha-1}}.$$

Proof. Recall that the dynamic energy consumption of an assignment $((p_i),(s_i))_{1\leq i\leq n}$ is $\sum_{i=1}^{n} p_i \times t_{i,p_i} \times s_i^{\alpha-1}$.

Let $t \in \mathbb{R}_+^*$. If $((p_i),(s_i))_{1\leq i\leq n}$ is an optimal assignment for the case $C_{\max} = 1$, with a total dynamic energy consumption of $E_{dyn}(1)$, then we can consider the same assignment but with all speeds divided by t, to ensure that all tasks meet the deadline $C_{\max} = t$: $((p_i),(\frac{s_i}{t}))_{1\leq i\leq n}$. The corresponding dynamic energy consumption is then $\frac{E_{dyn}(1)}{t^{\alpha-1}}$, and hence the optimal solution $E_{dyn}(t)$ is such that $E_{dyn}(t) \leq \frac{E_{dyn}(1)}{t^{\alpha-1}}$.

Conversely, if we have an assignment for the problem with $C_{\max} = t$, with a dynamic energy consumption of $E_{dyn}(t)$, then we take the same assignment but with all speeds multiplied by t to obtain a valid solution to the problem with $C_{\max} = 1$, hence leading to $E_{dyn}(t) \geq \frac{E_{dyn}(1)}{t^{\alpha-1}}$.

This concludes the proof of the lemma since $E_{dyn}(t) = \frac{E_{dyn}(1)}{t^{\alpha-1}}$ for any $t \in \mathbb{R}_+^*$. □

Theorem 10. *MinE-OneShelf-Cont can be solved optimally in polynomial time.*

Problem	Base ratio ($\mathcal{A}$)	Achieved bound	Result
MinE-Mold-Indep	OPT	OPT	Sec. 3.4.1
MinE-Rig-SS	c	c	Th. 4 (Sec. 3.5.1)
MinE-Mold-SS	$a\frac{W}{p} + bt_{\max}$	$a + b$	Th. 5 (Sec. 3.5.1)
	$\max(a\frac{W}{p}, bt_{\max})$	$\max(a, b)$	
MinE-Mold	$a\frac{W}{p} + bt_{\max}$	$a + b$	Th. 6 (Sec. 3.5.2)
	$\max(a\frac{W}{p}, bt_{\max})$	$\max(a, b+1)$	
MinE-Rig-Cont-SS	c	$c^{1-\frac{1}{\alpha}}$	Th. 7 (Sec. 3.6.1)
MinE-Mold-Cont-SS	$a\frac{W}{p} + bt_{\max}$	$(a+b)^{1-\frac{1}{\alpha}}$	Th. 8 (Sec. 3.6.2)
	$\max(a\frac{W}{p}, bt_{\max})$	$\max(a, b)^{1-\frac{1}{\alpha}}$	
MinE-OneShelf		Optimal	Th. 9 (Sec. 3.7.2)
MinE-OneShelf-Cont		Optimal	Th. 10 (Sec. 3.7.3)

Table 3.3: Summary of theoretical results from Section 3.4to Section 3.7. The base ratio is the approximation ratio on the makespan of the base algorithm $\mathcal{A}$.

Proof. The proof for a fixed $C_{\max}$ is the same as in the proof of Theorem 9, since we use the same dynamic programming algorithm. Then, $E_{n,p} = E_{dyn}(1) + p \times P_{stat}$. From Lemma 7, the optimal energy for a given $C_{\max}$ is therefore $E(C_{\max}) = E_{dyn}(C_{\max}) + E_{stat}(C_{\max}) = \frac{E_{n,p} - p \times P_{stat}}{C_{\max}^{\alpha-1}} + p \times C_{\max} \times P_{stat}$, which is a convex function of $C_{\max}$ that reaches its minimum for $C_{\max} = \sqrt[\alpha-2]{(\alpha-1)(\frac{E_{n,p}}{p \times P_{stat}} - 1)}$. The complexity of the algorithm is $O(np^2)$. □

The next section empirically assesses the theoretical results summarized in Table 3.3.

3.8 Empirical study

We first describe the experimental setup in Section 3.8.1. Then, we explain how instances are generated in Section 3.8.2. The different heuristics are compared and analyzed in Section 3.8.3, Also, we further study the impact of P_{stat} in Section 3.8.4, and the impact of having a set discrete speeds instead of the continuous model in Section 3.8.5.

3.8.1 Experimental setup

All the algorithms we use rely on the global mechanism presented in Algorithm 1 (Section 3.5.1 [**turek1992approximate**]) with a single speed (denoted with the suffix SS) and Algorithm 2 with multiple speeds (without any suffixes). The core idea is that first we transform a moldable instance into a rigid instance by fixing the number of processors for each task. It is then combined with the strategies presented in Section 3.2: ListBased and ShelfBased. The two algorithms ListBased and ShelfBased are

Algorithm 6: Algorithm ListBased for rigid tasks $\{(T_1, p_1, s_1), \ldots, (T_n, p_n, s_n)\}$

```
1  λ ← Empty schedule;
2  for j ∈ {1, ... p} do
3  |  C_j ← 0;
4  T ← {T_1, ..., T_n};
5  P ← ∅;
6  C_current ← 0;
7  while T ≠ ∅ do
8  |  if ∃T_i ∈ T s.t. p_i ≤ |P| then
9  |  |  i ← min_{T_i ∈ T} i s.t. p_i ≤ |P|;
10 |  |  λ ← λ ∪ {T_i starting at time C_current on p_i processors from P};
11 |  |  for k ∈ {1, p_i} do
12 |  |  |  Let j ∈ P;
13 |  |  |  C_j ← C_current + t_{i,p_i,s_i};
14 |  |  |  P ← P\{j};
15 |  |  T ← T\{T_i};
16 |  else
17 |  |  j ← arg min_{j ∈ {1,...,p}\P} C_j;
18 |  |  C_current ← C_j;
19 |  |  while C_current = min_{j ∈ {1,...,p}\P} C_j do
20 |  |  |  j ← min_{j ∈ {1,...,p}\P} C_j;
21 |  |  |  P ← P ∪ {j};
22 return the schedule λ;
```

Algorithm 7: Algorithm ShelfBased for rigid tasks $\{(T_1, p_1, s_1), \ldots, (T_n, p_n, s_n)\}$

Sort the tasks by non increasing execution time so that T_1 has the longest execution time and T_n the shortest;
$\lambda \leftarrow$ Empty schedule;
$C_{current} \leftarrow 0$;
$C_{next} \leftarrow 0$;
$\mathcal{T} \leftarrow \{T_1, \ldots, T_n\}$;
$\mathcal{P} \leftarrow \emptyset$;
for $T_i \in \mathcal{T}$ *by increasing* i **do**
    if $p_i > |\mathcal{P}|$ **then**
        $C_{current} \leftarrow C_{next}$;
        $C_{next} \leftarrow C_{current} + t_{i,p_i,s_i}$;
        $\mathcal{P} \leftarrow \{1, \ldots, p\}$;
    $\lambda \leftarrow \lambda \cup \{T_i$ starting at time $C_{current}$ on p_i processors from $\mathcal{P}\}$;
    for $k \in \{1, p_i\}$ **do**
        $\mathcal{P} \leftarrow \mathcal{P} \backslash \{j\}$;
return the schedule λ;

detailed as Algorithms 6 and 7. Moreover, we also implemented two optimization algorithms that can only be applied to an output of ShelfBased:

- OptiShelf, which optimizes each shelf once each task has been allocated to a shelf using the algorithm from Section 3.7 (this may change the number of processors used for each task). This optimization keeps the shelf structure, which can be an advantage for instances where this structure is a constraint the final schedule is subject to;

- De-shelf, which takes a ShelfBased solution and starts each task as soon as possible by removing the shelf constraint while keeping the allocations and the order in which the tasks are started.

For both of these optimizations, the energy consumption cannot be worse after the optimization than before. It is technically possible to combine the two optimizations (running OptiShelf and then De-shelf). We tried it for the sake of completeness, however this did not provide any interesting results as OptiShelf's optimization is heavily based on the shelf structure, while De-shelf removes this shelf structure. Overall, this represents a total of eight heuristics (two list-based, two non-optimized shelf-based, and four optimized versions of shelf-based).

To compare the different heuristics, we implemented them in C++17 compiled with `gcc` 9.3.0 with optimization option `-O3`. We rely on Python 3.8.5 to generate the instances and to analyze the results. The code of these experiments can be found on

Figshare[1].

3.8.2 Instance generation

The characteristics of the processors were extracted from a realistic platform [**moody10**, **benoit16**]:

Processor	p	P_{stat}	α	S
Intel Xscale	32	$\frac{6}{155} \approx 3.9 \times 10^{-2}$	3	$\{0.15, 0.4, 0.6, 0.8, 1\}$
Transmeta Crusoe	32	$\frac{44}{57560} \approx 7.6 \times 10^{-4}$	3	$\{0.45, 0.6, 0.8, 0.9, 1\}$

The number of tasks varies from 20 to 1000, with a step every 20 tasks. The workload was generated with the two following task profiles (half from each type):

- Amdahl's law [**amdahl1967**, **sun2018**]: $w_{i,p_i} = w_{i,1} \times \beta + \frac{w_{i,1} \times (1-\beta)}{p_i}$;
- Power law [**prasanna1996**, **hartstein2008**, **sun2018**]: $w_{i,p_i} = \frac{w_{i,1}}{p_i^{\beta}}$.

In both cases, $w_{i,1}$ and β are drawn from a uniform distribution $U(0, 1)$.

3.8.3 Results

In order to evaluate the performance of the various heuristics and show whether they return results close to the optimal, we compare the results with a lower bound that consists in an optimal execution in the MinE-Mold-Indep case (Section 3.4.1). In that case, the static energy is paid only while a task is executed, and any solution to MinE-Mold will consume at least as much energy as this lower bound. For convenience, the default version of a heuristic is the multiple-speed variant, and we refer to the single-speed variant with the SS suffix.

Figures 3.1 and 3.2 present an overview of the results for all heuristics with $n = 500$ tasks, respectively on the Intel Xscale platform and on the Transmeta Crusoe platform. We report both the ratio between the energy consumption of each heuristic with the lower bound (the lower the better), and also the execution time of the C++ implementation of the heuristics. A first remark is that single-speed and multiple-speed variants give very similar results. Indeed, in practice, we could confirm that the multiple-speed heuristics give the same speed to most tasks.

Figures 3.3 and 3.4 present a similar overview with a larger amount of tasks: $n = 5000$ tasks per instance. The difference in energy consumption between ListBased and De-shelf becomes smaller as n increases, while the execution time of the algorithm ListBased becomes much larger due to a higher order of growth.

Figures 3.5 and 3.6 present the scaling with n of all heuristics respectively on the Intel Xscale platform and the Transmeta Crusoe platform. With a large number of tasks,

[1] https://doi.org/10.6084/m9.figshare.14854395

the ratio with the lower bound becomes very close to 1 for all heuristics, at the price of an increasing execution time.

In terms of energy consumption, the best performing algorithms are ListBased and ListBased-SS (the lowest lines on the left charts). Then De-shelf and De-shelf-SS have a performance that is close to the ones of our best algorithms. Finally ShelfBased, ShelfBased-SS, OptiShelf and OptiShelf-SS have the worst performance among our algorithms (the top lines on the left charts).

In terms of execution time of the algorithms, most of our algorithms give instanta-

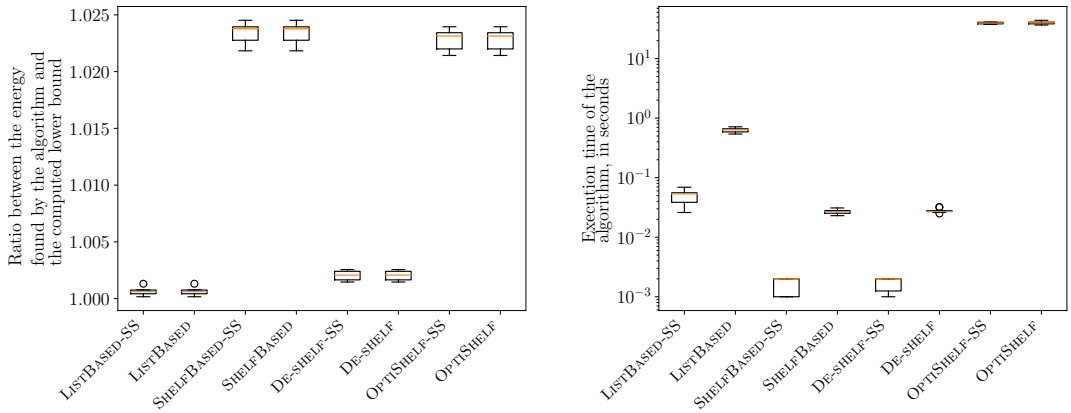


Figure 3.1: Output energy consumption and execution time to compute the solution for all eight heuristics with $n = 500$ mixed power and Amdahl's tasks and on the Intel Xscale platform ($p = 32$ processors with $P_{stat} = \frac{6}{155}$, $\alpha = 3$, $S = \{0.15, 0.4, 0.6, 0.8, 1\}$). Each box aggregates 10 measurements.

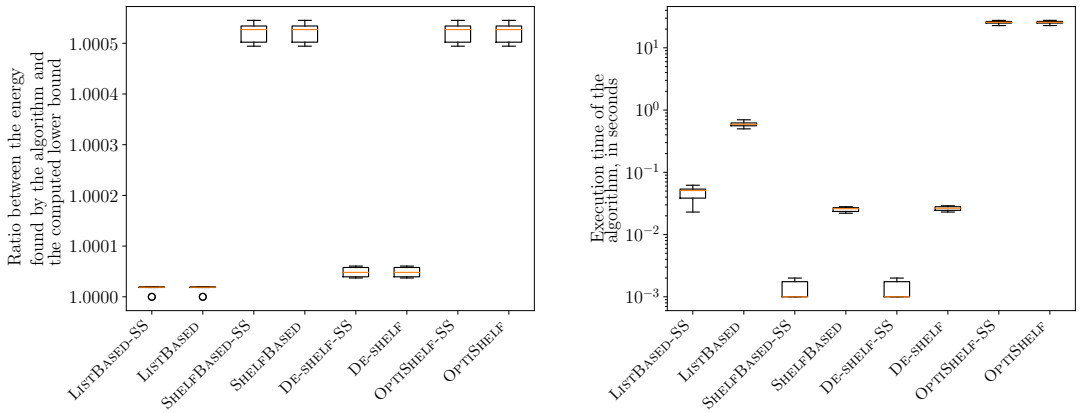


Figure 3.2: Output energy consumption and execution time to compute the solution for all eight heuristics with $n = 500$ mixed power and Amdahl's tasks and on the Transmeta Crusoe ($p = 32$ processors with $P_{stat} = \frac{44}{57560}$, $\alpha = 3$, $S = \{0.45, 0.6, 0.8, 0.9, 1\}$). Each box aggregates 10 measurements.

neous results. The exceptions are ListBased, that has a superlinear complexity with respect to the number of tasks, and both OptiShelf and OptiShelf-SS, that have a linear complexity with respect to the number of tasks but with a high constant factor.

As there are many algorithms and plots are overlapping, we then compare them in a more refined study, presenting more precisely how each algorithm behaves, along with the advantages and drawbacks of these algorithms.

Among the base algorithms (ListBased-SS, ListBased, ShelfBased-SS and ShelfBased), we focus on two baseline algorithms:

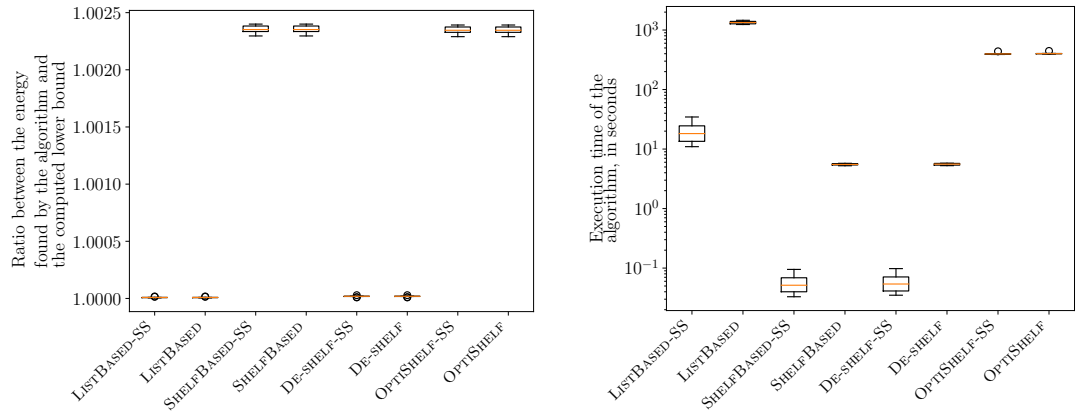


Figure 3.3: Output energy consumption and execution time to compute the solution for all eight heuristics with $n = 5000$ mixed power and Amdahl's tasks and on the Intel Xscale platform ($p = 32$ processors with $P_{stat} = \frac{6}{155}$, $\alpha = 3$, $S = \{0.15, 0.4, 0.6, 0.8, 1\}$). Each box aggregates 10 measurements.

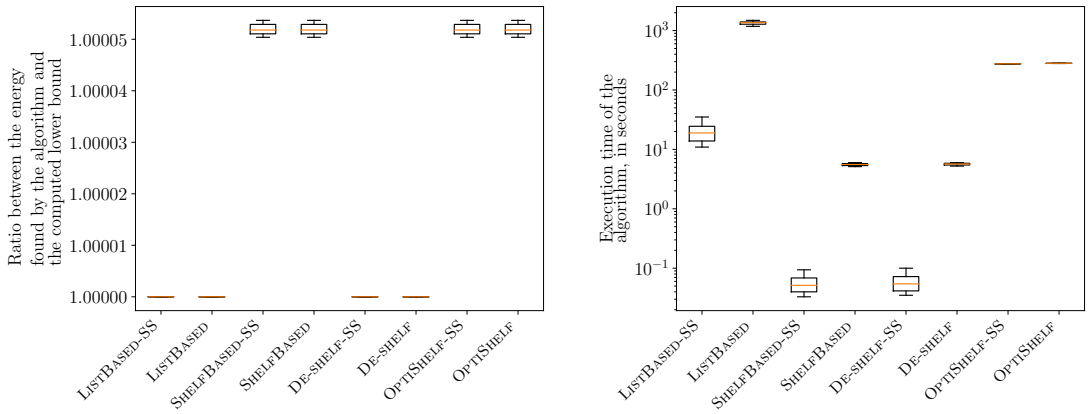


Figure 3.4: Output energy consumption and execution time to compute the solution for all eight heuristics with $n = 5000$ mixed power and Amdahl's tasks and on the Transmeta Crusoe ($p = 32$ processors with $P_{stat} = \frac{44}{57560}$, $\alpha = 3$, $S = \{0.45, 0.6, 0.8, 0.9, 1\}$). Each box aggregates 10 measurements.

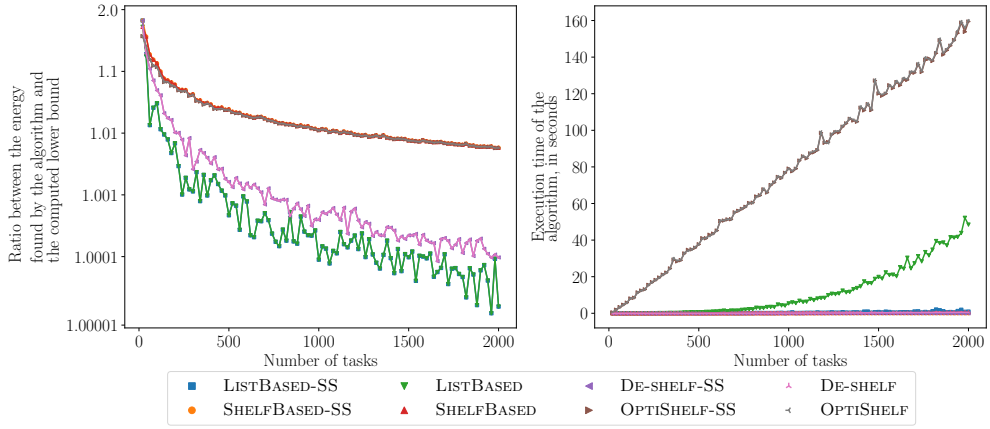


Figure 3.5: Output energy and execution time to compute the solution for all algorithms for instances with mixed power and Amdahl's tasks and and on the Intel Xscale platform ($p = 32$ processors with $P_{stat} = \frac{6}{155}$, $\alpha = 3$, $S = \{0.15, 0.4, 0.6, 0.8, 1\}$). The output energy is given with a $x \mapsto \log_{10}(x-1)$-scale.

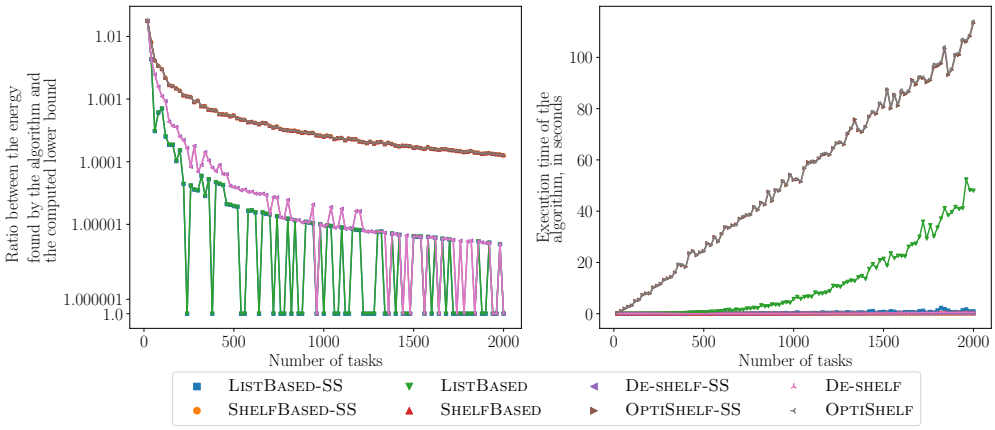


Figure 3.6: Output energy and execution time to compute the solution for all algorithms for instances with mixed power and Amdahl's tasks and and on the Transmeta Crusoe platform ($p = 32$ processors with $P_{stat} = \frac{44}{57560}$, $\alpha = 3$, $S = \{0.45, 0.6, 0.8, 0.9, 1\}$). The output energy is given with a $x \mapsto \log_{10}(x-1)$-scale.

- the base algorithm with the best output energy: ListBased;
- the base algorithm with the best execution time: ShelfBased-SS.

If we compare these two algorithms, ListBased and ShelfBased-SS, we can see that ListBased provides schedules with a lower energy consumption than ShelfBased-SS, but at the cost of a much larger execution time for the algorithm. When the number of tasks n increases, the difference in terms of execution time increases, while the difference of energy consumption decreases. Note that ListBased could be implemented in a faster way with a segment tree to compute which task can be started, making this operation $O(\log p)$ instead of $O(n)$. However, this complex data structure would probably not be included in most implementations.

However, we can use the solution delivered by ShelfBased (with a single speed for all tasks), and pass this solution through two possible optimizations: OptiShelf or De-shelf. By comparing the results of the two approaches, we can see that both optimizations increase the quality of the solution, but OptiShelf does it at the cost of a very large increase in the execution time. However, the overhead of De-shelf is small, which leads to solutions of better quality at a small cost.

Finally, we compare ListBased (with multiple speeds) to the optimized ShelfBased-SS (with a single speed) with De-shelf, which we found to be the best optimization for ShelfBased. As we can see on Figures 3.5 and 3.6, for very small instances, ListBased still performs better than ShelfBased with De-shelf. However, when n grows larger, ShelfBased with De-shelf quickly performs as well as ListBased, but with a lower time complexity.

Overall, by comparing the results we get with the processors Intel Xscale and Transmeta Crusoe, we see that, for all of the algorithms we provide, the schedules given for the Transmeta Crusoe are closer to the lower bound. First, this can be explained by the fact that the Transmeta Crusoe is a processor with a very low relative static power (around 7.6×10^{-4}) while the relative static power of the Intel Xscale is higher (around 3.9×10^{-2}). It means that there is less need to optimize the makespan, thus simplifying the problem. We explore the impact of P_{stat} in Section 3.8.4. Another explanation can be related to the difference of available speeds between the processors. We explore the impact of available speeds in the Section 3.8.5 through an empirical study of the continuous case.

3.8.4 Impact of P_{stat}

Figure 3.7 compares all the proposed algorithms when varying P_{stat}. We can observe that for small values of P_{stat}, even the algorithms that were not so efficient before provide good results. That is because, in this case, the idle time of the processors does not have a high cost in terms of energy consumption. Thus, having a small total amount of work W is more important than having a small makespan $C_{\max}$. Since all the algorithms try at some point to minimize W in the same way, they end up by all providing similar results.

However, when P_{stat} increases, the importance of minimizing the makespan $C_{\max}$ increases. The different algorithms provide different performance in terms of makespan,

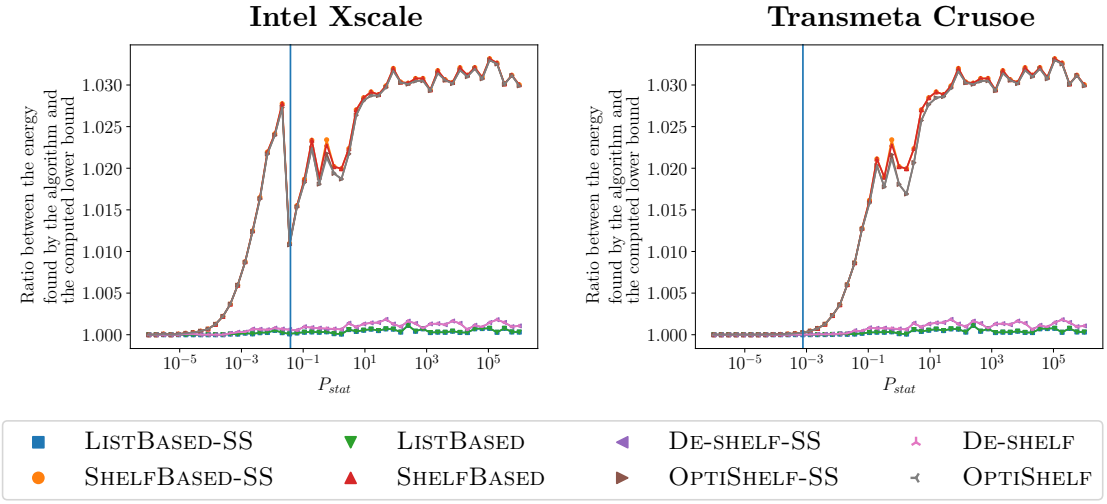


Figure 3.7: Output energy consumption to compute the solution for a variety of algorithms for instances with mixed power and Amdahl's tasks and $p = 32$ processors (on the left: with Intel Xscale $\alpha = 3$, and $S = \{0.15, 0.4, 0.6, 0.8, 1\}$; on the right: with Transmeta Crusoe $\alpha = 3$, and $S = \{0.45, 0.6, 0.8, 0.9, 1\}$). The vertical plain line corresponds to the actual P_{stat} of the studied processor.

which explains the difference in performance: ListBased and optimized ShelfBased algorithms provide much better solutions than simple ShelfBased algorithms.

3.8.5 Comparison with the continuous relaxation

Finally, we conduct experiments with the relaxed continuous version of the problem, MinE-Mold-Cont, where the speed of the processors can be any positive number. The previous lower bound does not apply for this relaxed problem because there is no constraint on the minimum speed. The ratio between the energy consumption achieved by the algorithm and the lower bound can thus be lower than 1. Note that we focus here on single speed variants of the algorithms, where a single continuous speed will hence be chosen.

Figure 3.8 compares all of the discrete speed algorithms we propose to the algorithm ListBased-Cont we use for the relaxed problem. This algorithm gives a solution with continuous speeds that consumes 15% less energy than the best result we can get with the speeds available for the Intel Xscale. It means that with a better choice of speeds when designing the processor, we can expect a 15% decrease in energy consumption with our algorithms. For the Transmeta Crusoe processor, the energy gap is even bigger: with a better choice of speeds, we can hope to gain more than 80% of energy.

Figure 3.9 compares the discrete-speed algorithm with the best results (ListBased-SS) to the result we can get with continuous speeds, for different values of P_{stat}. Intuitively, all approaches are close to 1 when they rely on speeds that are close to the

discrete speeds of the studied processor: in these cases the speeds used by the continuous speed algorithm are already available in the discrete speed model. When we get further away from these cases, we see that allowing continuous speeds would allow for a much better performance. It means that the design of the static power P_{stat} and the set of available speeds S must be done concordantly. This design can be helped by theoretical results, such as the ones we provide in this chapter, along with simulations such as the ones we provide in this section.

When comparing the Intel Xscale, on the left, to the Transmeta Crusoe, on the right, we see that the available speeds for the Intel Xscale correspond well to the effective P_{stat} (the plain line). It is not the case for the Transmeta Crusoe: lower processor speeds would improve the energy consumption.

3.9 Conclusion

With the growing concern regarding the energy consumption of current parallel platforms, it is crucial to bound the worst-case performance. This work is the first to propose such bounds on the energy consumption when scheduling moldable tasks. We highlight the relation between the energy and the completion time (determined by the DVFS mechanism) and rely on the numerous approximation algorithms that have already been proposed to minimize the completion time. This leads to a general mechanism to bound the energy consumption of such existing approximation algorithms for the completion time. In particular, we show that a shelf-based approach is a 3-approximation (resp.

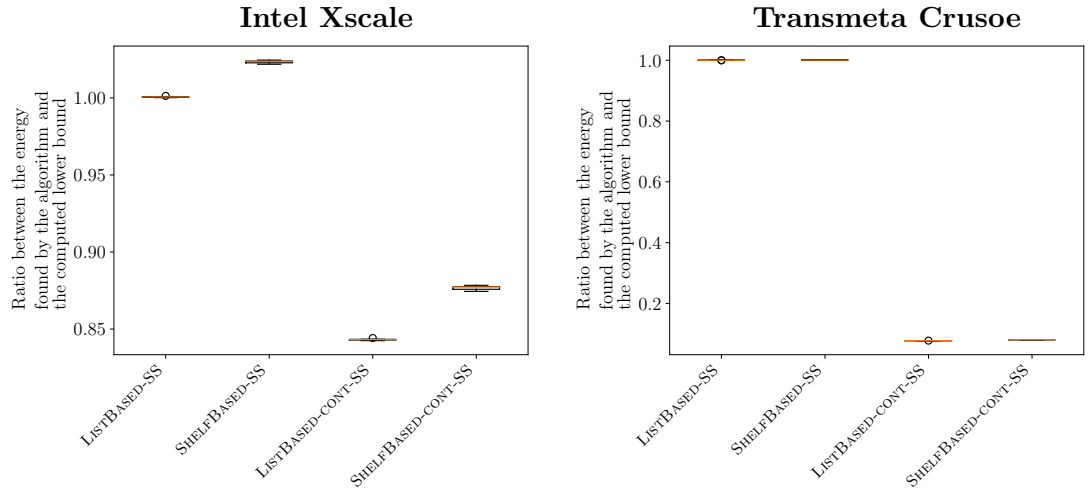


Figure 3.8: Output energy consumption and execution time to compute the solution for four heuristics with $n = 500$ mixed power and Amdahl's tasks and $p = 32$ processors (on the left: with Intel Xscale $P_{stat} = \frac{6}{155}$, $\alpha = 3$, and $S = \{0.15, 0.4, 0.6, 0.8, 1\}$ in the discrete cases; on the right: with Transmeta Crusoe $P_{stat} = \frac{44}{57560}$, $\alpha = 3$, and $S = \{0.45, 0.6, 0.8, 0.9, 1\}$ in the discrete cases). Each box aggregates 10 measurements.

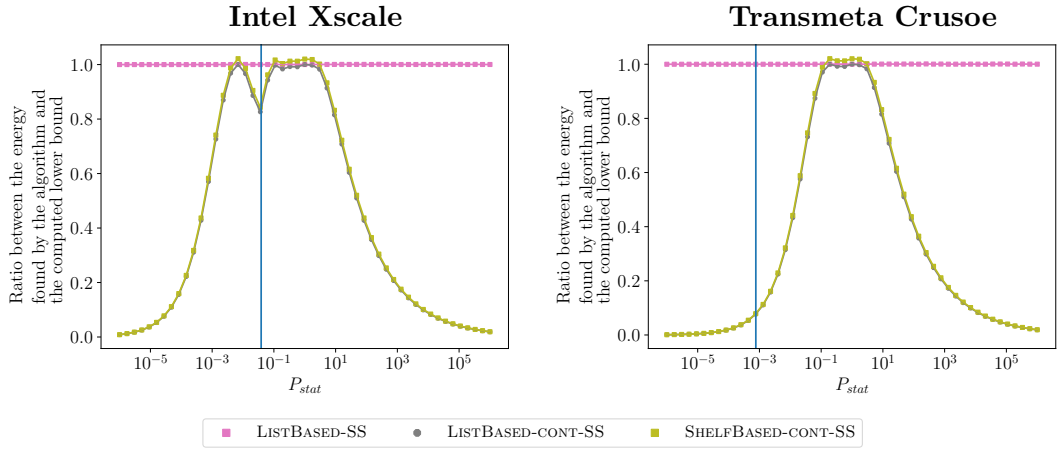


Figure 3.9: Output energy consumption to compute the solution with discrete and continuous speeds for instances with mixed power and Amdahl's tasks and $p = 32$ processors (on the left: with Intel Xscale $\alpha = 3$, and $S = \{0.15, 0.4, 0.6, 0.8, 1\}$ in the discrete cases; on the right: with Transmeta Crusoe $\alpha = 3$, and $S = \{0.45, 0.6, 0.8, 0.9, 1\}$ in the discrete cases). The vertical plain line corresponds to the actual P_{stat} of the studied processor.

2.08-approximation) algorithm for the energy consumption with discrete speeds (resp. continuous speeds).

We also focus on the optimization of a single shelf by providing a polynomial-time algorithm that can be used to improve existing solutions. Empirical results reveal that such an approach, when combined with a fast optimization post-operation, is beneficial in practice because of its low cost.

To complete this study, we could consider variations of the power model. In particular, we assume that changing frequencies with the DVFS mechanism does not incur any energy cost or delay, which may not be accurate in practice. We additionally assume that the set of available speeds is a constant, while we could consider it a function of the number of processors in use (e.g., when all the processors are used, the highest frequencies might not be usable). Also, we plan to experiment on more recent processors whose range of frequencies suggest that it behaves closer to the continuous model. This involves doing some measurements to obtain detailed data on the power consumption of such processors and processing these measures, while only information on frequencies is publicly available. Finally, the whole model revolves on the premise that the tasks are computationally bound. In the case of memory bound applications, changing the processor speeds does not change the execution time, which is a case that we have not considered yet. The algorithms presented could be adapted to perform on mixes of computationally bound and memory bound applications.

Overall, this work aims at providing the theoretical foundations to the problem. Since current task systems do not yet have moldable task profiles that can be used, we have focused on classical models that have already been largely considered in the

literature. As soon as moldable task profiles are available, it would be very interesting to conduct experiments on real HPC systems.

Chapter 4

Asymptotic performance and energy consumption of SLACK

4.1 Introduction

The problem of minimizing the computation time when scheduling n independent tasks on m identical processors is at the basis of scheduling theory, and a building block for solving many more complicated problems, hence it remains very important even though it has already been widely studied. Using Graham's notation [**graham1979**], this problem is denoted $P||C_{\max}$.

While the problem is NP-complete (equivalent to 2-partition with two processors, or 3-partition when the number of processors m is part of the input), an easy way to get efficient solutions consist in ordering the n tasks according to some criterion, and then perform a list schedule, i.e., schedule the next task of the list on the least loaded processor, hence never leaving a processor idle. A classic ordering is the one of LPT (Longest Processing Time), which orders tasks from the longest to the smallest [**graham1969**]. This algorithm has proven to have good theoretical and even better practical performance. In particular, its rate of convergence has been studied, and new results were recently established when the distribution of task costs is generated using uniform integer compositions [**benoit2021**].

More recently, the SLACK heuristic was proposed in [**della2020**], showing promising empirical performance compared to LPT. Its principle is based on grouping tasks of similar execution times into packs, sorting the resulting packs by non-decreasing similarity (the similarity of a pack denoting the maximum difference of execution times between its tasks), and then scheduling the tasks in the order determined by the packs, following a list schedule (assign the next task to the least loaded processor). The idea is that a single pack cannot bring the imbalance of the processors too high, and the hope is that the packs balance each other. The objective is that the tasks in the last scheduled packs are very close to each other, hence they will not create a large imbalance at the end of the schedule. While this SLACK algorithm benefits from favorable empirical performance, fewer analyses have been conducted on its theoretical properties.

These heuristics were proposed in order to minimize the makespan, i.e., the maximum execution time among the processors. Another core problem consists in minimizing the *energy consumption*, as the energy consumption of current platforms is an ever-growing concern, both for economical and ecological reasons. To optimize the energy consumption, modern processors can run at different speeds, and their power consumption is then the sum of a static part (the cost for a processor to be turned on) and a dynamic part, which is a strictly convex function of the processor speed. More precisely, a processor running at speed s dissipates a power of s^α Watts, where $2 \leq \alpha \leq 3$ [**bambagini2016**]. Hence, a higher speed allows executing a task more rapidly, but at the price of a much higher amount of energy consumed. Finding a schedule now consists in deciding on which processor to execute each task and to decide at which speed the task is executed.

Therefore, we revisit this classic problem of scheduling n independent tasks onto m identical processors, with the aim of deriving analytical results for SLACK, when the goal is to minimize the makespan or the energy consumption. We study the performance of SLACK from an asymptotical point of view, under the assumption that the execution times of the tasks follow a given probability distribution. The study is building on a comparison of the most heavily loaded machine compared to the least loaded one, and hence it provides interesting insights both for the study of the classic makespan objective function, and its translation to the energy consumption. The goal of this chapter is therefore to answer two main questions left unresolved in the literature so far: (i) provide a theoretical study to analyze the performance of SLACK, and (ii) consider the energy consumption in the theoretical and empirical analysis of the algorithms. Our main contributions are the following:

- A fundamental bound related to the result of SLACK (Section 4.4);
- A convergence rate for the makespan of SLACK when using uniform and exponential distributions, by applying the bound of Section 4.4 (Section 4.5.1);
- A general result for bounding the energy consumption (agnostic of the algorithm and the task distribution) and its application to SLACK, by applying the bound of Section 4.4 (Section 4.5.2);
- Simulations for comparison with the theoretical bounds that were computed for SLACK and LPT (Section 4.6).

First, Section 4.2 summarizes the existing contributions related to either the energy minimization problem or LPT and SLACK. Section 4.3 presents the problems and algorithms (LPT and SLACK). Then, Section 4.4 presents a useful bound on the result given by SLACK. Section 4.5 proposes applications of this bound: theoretical asymptotic results related to the minimization of the makespan and the energy with SLACK. In the case of the energy, Section 4.5.2 also gives a method to derive energy related guarantees for any algorithm bounded similarly to SLACK in Section 4.4. Section 4.6 presents the experimental results of the empirical study of LPT and SLACK. Finally, Section 4.7 concludes.

4.2 Related work

Lowering the energy consumption of computational tasks has been widely studied in the last decades, be it in the context of High Performance Computing or in other contexts, such as Cloud Computing. Many models have been proposed for the energy consumption of CPUs. For instance, the energy consumption is scaling quadratically with the speed of the CPU in [**weiser1994**], and there is a focus on the online evaluation of the expected idle time. In [**yao1995**], the only assumption is that the energy consumption is a convex function of the speed of the CPU, and clairvoyant online and offline solutions are proposed to the problem. The heuristics presented in these two articles are then evaluated, either empirically in [**weiser1994**], or with approximation ratios in [**yao1995**]. In our work, we explore another way of evaluating algorithms, following the remark that with large systems, stochastic asymptotic results should be relevant.

Recent surveys such as [**czarnul2019**] and [**thakkar2020**] compile various techniques used for energy-efficient computing, including scheduling techniques. These techniques may use either Dynamic Voltage and Frequency Scaling (DVFS), as in [**lin2014**], where the frequency (and hence the speed) of processors may be chosen, or Dynamic Power Management (DPM) as in [**benini2000**]. These studies propose algorithms, but they mainly focus on an empirical evaluation of these algorithms, without theoretical study.

As for scheduling algorithms that have low complexities (and therefore low energy consumption), LPT has been a well known algorithm for decades and is known to provide good theoretical and practical performance while keeping a low time complexity in $O(n \log n)$ [**graham1969**]. A more recent algorithm, SLACK, also remains with an $O(n \log n)$ time complexity, while providing results that are sometimes better than LPT [**della2020**, **benoit2021**].

There are multiple results about the asymptotic behavior of LPT under different assumptions. Frenk and Rhinnooy Kan [**frenk1986**] and Coffman et al. [**coffman1988**] study the difference between LPT and the optimal solution in the case where the execution times of the tasks follow a probability distribution of cumulative distribution function of the form $F(x) = x^\alpha$, where $0 < \alpha < +\infty$. Loulou [**loulou1984**] and Piersma and Romeijn [**piersma1996**] do not look at specific distributions, but instead they study LPT under the assumption that the execution times are independent and identically distributed random variables. More recently, Benoit et al. [**benoit2021**] studied the asymptotic optimality of SLACK and LPT under the assumption that the execution times are generated using a distribution called the uniform integer composition.

4.3 Framework

The $P||C_{\max}$ problem is a classic scheduling problem, where n tasks have to be scheduled on m identical machines, with the objective function of makespan minimization, i.e., minimize the execution time of the machine that completes last ($C_{\max}$). There are no constraints on tasks, which can be assigned to any machine in any order. Each task has

a number of operations to perform, that we call its work and denote by w_i, and the time to execute the task is usually $t_i = w_i$, assuming that the machine executes one operation per time unit (speed $s = 1$). The problem complexity is well known, and in particular the associated decision problem is NP-complete as soon as $m \geq 2$.

List scheduling and LPT. In order to solve this $P||C_{\max}$ problem, a simple but effective heuristic algorithm consists in never letting a machine idle, i.e., as soon as a task completes on a machine, a new task is assigned to this machine. This is called *list scheduling*, and it can be implemented as in Algorithm 8, by keeping the load of each machine in a vector $\overrightarrow{W}$ of length m initialized to $(0, 0, \ldots, 0)$. For each task, we assign it to the currently least loaded machine, and the makespan is the maximum value of the vector $\overrightarrow{W}$ at the end of the execution. Any list schedule (whatever the order of tasks) is know to be a $(2 - \frac{1}{m})$-approximation algorithm [**graham1969**]. A variant of the List Scheduling heuristic consists in first sorting the list L by non-increasing task works, and it is called *Longest-Processing-Time-first* (LPT for short). This can be used if all tasks are known beforehand (offline scheduling), and it improves the approximation ratio of the algorithm to $(\frac{4}{3} - \frac{1}{3m})$ [**graham1969**].

SLACK**.** In this chapter, we mainly focus on the SLACK algorithm, that was introduced in [**della2020**] and consists in applying the List Scheduling heuristic with a particular pretreatment on the list of tasks, as detailed in Algorithm 9. We first fill the list L to have a number of elements r that is a multiple of m, by adding dummy tasks of work 0. Then, tasks are sorted by non-increasing works and grouped by packs of m tasks, and then the packs are themselves sorted by non-increasing difference between the work of the longest task of the pack and the smallest one (α_i's). These differences are denoted β_k, where $\beta_1 \geq \beta_2 \ldots \geq \beta_{r/m}$. They correspond to the sorted α_i's.

Let us denote by $c_i(j)$ the load of processor j after $i \times m$ tasks (i.e., the i first packs) have been scheduled. Hence, $c_i(j) = \overrightarrow{W}[j]$ after $i \times m$ steps of the loop line 2 of Algorithm 8. One has for instance $c_0(j) = 0$ for all j (initial load), and then at each iteration i, we schedule one more pack with m tasks. We then define $\delta_i = \max_{0 \leq j,j' < m}(|c_i(j) - c_i(j')|)$, which is the maximum difference of load between two processors after iteration i.

Note that these values β_i and δ_i can be extended to any algorithm, in particular LPT, by simply grouping tasks by *packs* of m tasks (in the order in which they are scheduled),

Algorithm 8: ListScheduling(L, m)

Require: List L of n positive floats (task works); Number of processors m.

1: Let $\overrightarrow{W}$ be a vector of length m initialized to $\overrightarrow{W} = (0, 0, \ldots, 0)$; **for** $w \in L$ *in the order they appear in the list* **do**
2: Let j be the index of a minimal element of $\overrightarrow{W}$;
3: $\overrightarrow{W}[j] = \overrightarrow{W}[j] + w$;
4: **return** $\overrightarrow{W}$;

Algorithm 9: SLACK (L, m)

Require: List L of n positive floats (task works); Number of processors $m \leq n$.

1: Add $(-n \mod m)$ elements of work 0 at the end of L;
2: $r = n + (-n \mod m)$;
3: $L' = [x_1, \ldots, x_r]$ is obtained by sorting L non-increasingly; **for** $0 \leq i \leq \frac{r}{m} - 1$ **do**
4: $K_i = [x_{im+1}, x_{im+2}, \ldots, x_{im+m}]$;
5: $\alpha_i = x_{im+1} - x_{im+m}$;
6: Let $H = [\alpha_{i_1}, \ldots, \alpha_{i_{\frac{r}{m}}}] = [\beta_1, \ldots, \beta_{\frac{r}{m}}]$ be a non-increasing sequencing of the α_i's; L_{SLACK} is obtained by concatenating the K_i's in the same order as the α's in H.
7: $\overrightarrow{W}$ =ListScheduling(L_{SLACK}, m);

and then checking differences between tasks of a same pack (β_i), and differences in the load of processors at each iteration, hence, after m tasks have been scheduled (δ_i).

From makespan to energy consumption. When the goal is to minimize the energy consumption, we further consider that the frequency of the processors can be scaled using DVFS (Dynamic Voltage and Frequency Scaling). Hence, these processors have a static power P_{stat}, and can be operated at any speed (or frequency) $s \in \mathbb{R}_+^*$ [**BKP07**], while we assumed so far that $s = 1$.

The execution time of task T_i at speed s then becomes $t_{i,s} = \frac{w_i}{s}$. In terms of energy consumption, there is a static part, which corresponds to the power consumed when the m processors are turned on, during a time $C_{\max}$, hence a total of $m \times C_{\max} \times P_{stat}$. For each task T_i, there is also a dynamic energy consumption, directly related to the speed s at which the processor operates the task. Using a general model, the dynamic energy consumption is $t_{i,s} \times s^\alpha$ [**bambagini2016**], where $\alpha > 1$ (in general, $2 \leq \alpha \leq 3$). Finally, the total energy consumption of a schedule of length $C_{\max}$, where T_i is operated at speed s_i, is:

$$E = m \times C_{\max} \times P_{stat} + \sum_{i=1}^{n} t_{i,s_i} \times s_i^\alpha.$$

For convenience, the main notations are summarized in Table 4.1.

4.4 A bound for SLACK

This section is dedicated to proving a fundamental bound related to the algorithm SLACK.

Let $\mathcal{X}$ be a distribution with positive values. We denote by $C(n, m, \mathcal{X})$ the random variable of the makespan returned by the SLACK algorithm on m processors on a list of n tasks that are independent random variables of distribution $\mathcal{X}$. Let $X_1, \ldots, X_n$ be n independent random variables distributed according to $\mathcal{X}$. Let $X_{1:n} \leq X_{2:n} \leq \ldots \leq X_{n:n}$ be associated order statistics. Particularly $X_{1:n}$ is the minimum of the X_i's and $X_{n:n}$ the maximum. Let $D_i = (X_{i:n} - X_{i-1:n})$ for every $1 \leq i \leq n$, with the convention $X_{0:n} = 0$. The D_i's are classically called spacings of adjacent order statistics. Let $\Delta_{\mathcal{X},n}$ be the

Symbol	Definition
m	number of processor
n	number of tasks
$\{T_1, \ldots, T_n\}$	the n tasks
w_i	work of T_i (corresponding to the number of operations required by the task)
$t_i = w_i$	the execution time of T_i at speed 1
$t_{i,s} = \frac{w_i}{s}$	execution time of T_i at speed s
$m \times C_{\max} \times P_{stat}$	static energy consumption for a duration $C_{\max}$
$t_{i,s} \times s^\alpha$	dynamic energy consumption of T_i at speed s
δ_i	largest difference between the total execution times of two processors after having processed $i \times m$ tasks (first i packs)
β_i	largest difference between the execution time of any two tasks in pack i
$c_i(j)$	total execution time of processor j after $i \times m$ tasks have been scheduled (first i packs)
$W_j = \sum_{alloc(i)=j} w_i$	total work (number of operations) on processor j; $alloc(i)$ is the processor on which T_i is allocated
$W_{\max} = \max_{1 \le j \le m} W_j$	maximal number of operations allocated to a processor
$W = \sum_{1 \le i \le n} w_i$	total number of operations to perform
$\overrightarrow{W} = (W_1, \ldots, W_m)$	the $\rightarrow$ notation is used for m-length vectors (not only for W)
$\|\overrightarrow{x}\|_\alpha = \sqrt[\alpha]{\sum x_i^\alpha}$	classic α-norm of a vector

Table 4.1: Main Notations

random variable of the maximal value of D_i's, that is the maximal difference between two consecutive $X_{i:n}$ (and between 0 and $X_{1:n}$).

Theorem 11. *When using* SLACK *(Algorithm 9),* $\max_{1 \le i,j \le m} (W_i - W_j) \le m\Delta_{\mathcal{X},n}$.

Lemma 8. *If at step $j+1$, we put two or more tasks on a processor, then each of these tasks has an execution time at most δ_j.*

Proof. Assume there is a processor a that receives two or more tasks at step $j+1$. It means that the processor b that initially had the highest execution time does not receive any task, otherwise it would mean that we have distributed exactly one task to each processor. We then get that before receiving its second task, the processor a has a lower current execution time than processor b, which means that the first task allocated to processor a at this step has an execution time at most the initial difference between

these two processors. So, this execution time is at most δ_j, and since the tasks are sorted in a step, all the subsequent tasks added to the processor a this step also fulfill this condition. □

Lemma 9. *For any processors a and b that both receive at least one task at step $j+1$, we have $|(c_j(a)+\tau_{a,1})-(c_j(b)+\tau_{b,1})| \leq \max(\delta_j, \beta_j)$, where $\tau_{a,1}$ (resp. $\tau_{b,1}$) is the execution time of the first task received by a (resp. by b).*

Proof. Assume without loss of generality that processor b receives its first task before processor a. Then, we have $c_j(b) \leq c_j(a)$ and $\tau_{a,1} \leq \tau_{b,1}$. Let $\tau_{\text{diff}} = \tau_{b,1} - \tau_{a,1}$. We have:

$$\begin{aligned}|(c_j(a)+\tau_{a,1})-(c_j(b)+\tau_{b,1})| &= |(c_j(a)-c_j(b))-\tau_{\text{diff}}| \\ &\leq \max((c_j(a)-c_j(b))-\tau_{\text{diff}}, \tau_{\text{diff}}-(c_j(a)-c_j(b))).\end{aligned}$$

Since $0 \leq \tau_{\text{diff}}$ and $0 \leq (c_j(a)-c_j(b))$ we have

$$\begin{aligned}|(c_j(a)+\tau_{a,1})-(c_j(b)+\tau_{b,1})| &\leq \max((c_j(a)-c_j(b)), \tau_{\text{diff}}) \\ &\leq \max(\delta_j, \beta_j).\end{aligned}$$

□

Lemma 10. *For every j, $\delta_{j+1} \leq \max(\beta_{j+1}, \delta_j)$.*

Proof. Let $p_{\min}$ be the processor minimizing c_{j+1} and $p_{\max}$ be the processor maximizing c_{j+1}. Let $\hat{\tau}_m = c_j(p_{\min})$ and $\hat{\tau}_M = c_j(p_{\max})$ be their respective execution times before adding the new tasks, and $\{\tau_{m,k}\}_{k \leq K_m}$ and $\{\tau_{M,k}\}_{k \leq K_M}$ are the execution times of the tasks allocated to these processors during this step (sorted by non increasing execution times).

By definition of $p_{\min}$ and $p_{\max}$, $\delta_{j+1} = c_{j+1}(p_{\max}) - c_{j+1}(p_{\min}) = (\hat{\tau}_M + \sum \tau_{M,k}) - (\hat{\tau}_m + \sum \tau_{m,k})$. We see different cases depending on the number of tasks on $p_{\max}$, denoted by K_M.

- Case 1: $K_M = 0$. In this case, the maximum processor has not changed and has not received any task this step, so the difference between the processor $p_{\max}$ and the other processors has only reduced at this step. So, we get $\delta_{j+1} \leq \delta_j \leq \max(\beta_{j+1}, \delta_j)$.
- Case 2: $K_M = 1$. In this case, we have:

$$\begin{aligned}\delta_{j+1} &= (\hat{\tau}_M + \tau_{M,1}) - (\hat{\tau}_m + \sum \tau_{m,k}) & \\ &\leq (\hat{\tau}_M + \tau_{M,1}) - (\hat{\tau}_m + \tau_{m,1}) & t_{m,k} \geq 0 \\ &\leq |(\hat{\tau}_M + \tau_{M,1}) - (\hat{\tau}_m + \tau_{m,1})| & \text{the quantity is already positive} \\ &\leq \max(\beta_{j+1}, \delta_j) & \text{Lemma 9}\end{aligned}$$

- Case 3: $K_M > 1$. By using Lemma 8, the last task added to processor $p_{\max}$ has size at most δ_j. The last task of $p_{\max}$ would have been added to any processor with lower execution time if possible, so at the end of step $j+1$, any processor has a total execution time at least:

$$\begin{aligned} T &\geq (\hat{\tau}_M + \sum \tau_{M,k}) - \tau_{M,K_M} \\ &\geq (\hat{\tau}_M + \sum \tau_{M,k}) - \delta_j && \tau_{M,K_M} \leq \delta_j \end{aligned}$$

So we get:

$$\begin{aligned} \delta_{j+1} &= (\hat{\tau}_M + \sum \tau_{M,k}) - (\hat{\tau}_m + \sum \tau_{m,k}) \\ &\leq (\hat{\tau}_M + \sum \tau_{M,k}) - ((\hat{\tau}_M + \sum \tau_{M,k}) - \delta_j) \\ &\leq \delta_j \\ &\leq \max(\beta_{j+1}, \delta_j) \end{aligned}$$

So, in all cases, we get $\delta_{j+1} \leq \max(\beta_{j+1}, \delta_j)$, proving the lemma. □

Lemma 11. *When using* SLACK *(Algorithm 9), for every j, $\delta_j \leq \beta_1$.*

Proof. The proof is done by induction on j. For the first iteration ($j = 1$), since the m first considered tasks are each assigned to a different processor, we have:

$$\begin{aligned} \delta_1 &= \max_{p,p'}(|c_1(p) - c_1(p')|) \\ &= \max_{k,k' \in [1,m]}(|x_{k+i_1 \times r/m} - x_{k'+i_1 \times r/m}|) && \text{replacing with the first tasks} \\ &= \alpha_{i_1} = \beta_1. \end{aligned}$$

Assume now that $\delta_j \leq \beta_1$ for a $j < r/m$. Using Lemma 10, we can bound $\delta_{j+1} \leq \max(\beta_{j+1}, \delta_j)$. Therefore, $\delta_{j+1} \leq \max(\beta_{j+1}, \beta_1)$. Since $\beta_1 \geq \beta_{j+1}$, we have $\delta_{j+1} \leq \beta_1$, which concludes the proof. □

We are now ready to prove Theorem 11.

Proof. We prove an upper bound for β_1.

$$\begin{aligned} \beta_1 &= \alpha_{i_1} \\ &= \max_{1 \leq i \leq m} (x_{1+i \times r/m} - x_{m+i \times r/m}) \\ &= \max_{1 \leq i \leq m} \left(x_{1+i \times r/m} - x_{m+i \times r/m} - \sum_{i=2}^{m-1} x_{1+i \times r/m} + \sum x_{1+i \times r/m} \right) \\ &= \max_{1 \leq i \leq m} \left(\sum_{i=1}^{m-1} x_{1+i \times r/m} - x_{1+(i+1) \times r/m} \right) \\ &\leq (m-1) \times \max_{1 \leq i \leq r-1} X_{i+1:n} - X_{i:n} \end{aligned}$$

$$\leq (m-1) \times \Delta_{\mathcal{X},n}$$

We just proved that

$$\beta_1 \leq (m-1) \times \Delta_{\mathcal{X},n}. \tag{4.1}$$

We conclude the proof by combining Equation (4.1) and Lemma 11 (since $\delta_n = \max_{1 \leq i,j \leq m} (W_i - W_j)$). □

4.5 Convergence speed of SLACK

In this section, we use the fundamental bound found in Section 4.4 to derive asymptotic results on the optimality of SLACK, first in terms of makespan in Section 4.5.1, and then in terms of energy consumption in Section 4.5.2.

4.5.1 Convergence of the makespan

This section is dedicated to prove asymptotic results on the optimality of SLACK. The following main result is a direct application of Theorem 11:

Proposition 1. *The makespan of* SLACK *differs from the optimal one by at most* $m\Delta_{\mathcal{X},n}$*:*

$$0 \leq C(n,m,\mathcal{X}) - OPT \leq \frac{(m-1)^2}{m}\Delta_{\mathcal{X},n} \leq m\Delta_{\mathcal{X},n}.$$

Proof. Since $OPT \geq \frac{1}{m}\sum_{i=1}^{r} x_i$, one has

$$\begin{aligned} m \times C(n,m,\mathcal{X}) &\leq (m-1) \times \delta_n + \sum_{i=1}^{r} x_i \\ &\leq (m-1) \times \delta_n + m \times OPT. \end{aligned}$$

Consequently (by Theorem 11),

$$0 \leq C(n,m,\mathcal{X}) - \mathrm{OPT} \leq \frac{(m-1)^2}{m}\Delta_{\mathcal{X},n} \leq m\Delta_{\mathcal{X},n}.$$

□

It is worth noting that for many bounded distributions, Proposition 1 will provide results on the convergence of the absolute error $(C(n,m,\mathcal{X}) - \mathrm{OPT})$ when n goes to infinity, as $\Delta_{\mathcal{X},n}$ is smaller when that execution times of the tasks gets denser.

Now, we will use known results on order spacings to obtain convergence results for SLACK. It is proved in [**https://doi.org/10.48550/arxiv.1909.06406**], [**Bairamov2010**] that

$$\mathbb{E}\left(\Delta_{\mathcal{U}[0,1],n}\right) \sim \frac{\ln n}{n+1}, \tag{4.2}$$

where $\mathcal{U}[0,1]$ is the uniform distribution between 0 and 1.

From Proposition 1 and Equation (4.2), one has the following result, proving that for a fixed m, the SLACK algorithm provides a scheduling that converges in expectation to the optimal (for the makespan):

Corollary 3. *For any fixed $m \geq 2$,*

$$0 \leq \mathbb{E}\left(C(n, m, \mathcal{U}[0,1])\right) - \mathbb{E}\left(OPT\right) = O\left(m\frac{\ln n}{n+1}\right).$$

A similar result can be obtained for the exponential distribution. It is shown in [**devroye1984exponential**] that, almost surely,

$$\limsup_{n\to+\infty}\left(\frac{\Delta_{\mathcal{E}_1,n}}{\ln\ln n}\right) = 1, \tag{4.3}$$

where $\mathcal{E}_1$ is the exponential distribution (with rate 1).

Using Proposition 1 and Equation (4.3), we then have the following result:

Corollary 4. *For any fixed $m \geq 2$, one has almost surely*

$$0 \leq \limsup_{n\to+\infty}\left(\frac{C(n, m, \mathcal{E}_1) - OPT}{m\ln\ln n}\right) \leq 1.$$

Corollary 4 does not show a convergence of the makespan of SLACK to the optimal, but that, almost surely, the gap between their difference is under control since $\ln\ln n$ has a very slow growing speed.

4.5.2 Convergence of the energy consumption

Building upon the previous results bounding the δ_i's for SLACK and analyzing its impact on the makespan, we now move to the problem of minimizing the total energy consumption E, where the speed of each processor can take any value in $\mathbb{R}_+^*$. The main result, stated in Theorem 12, shows how to adapt a classic scheduling algorithm (without speed and energy consideration) into an energy-oriented one. The quality of the solution is bounded by a factor depending on the maximal difference δ between the execution times of the last finishing processor and the first finishing processor.

We start with a preliminary lemma that further defines the shape of optimal solutions: each processor has a constant speed and all processors finish at the same time.

Lemma 12. *In an optimal solution, each processor has a constant speed, and all processors finish at the same time.*

Proof. We first prove that each processor has a constant speed, and then that all processors finish at the same time.

Each processor has a constant speed. Let us assume that there is a processor that does not have constant speed. It means that there are two consecutive amount of work w_1, w_2 being processed at different speeds s_1, s_2. Let $\gamma_1 = \frac{1}{s_1}$, $\gamma_2 = \frac{1}{s_2}$, $\gamma = \frac{w_1 \times \gamma_1 + w_2 \times \gamma_2}{w_1 + w_2}$. Notice that γ is the weighted average of γ_1 and γ_2 (i.e., $\gamma = t \times \gamma_1 + (1-t) \times \gamma_2$ with $t = \frac{w_1}{w_1+w_2}$). By running both amounts of works at speed $\frac{1}{\gamma}$, the total execution time does not change, so the static energy does not change. By strict convexity of $f(x) \mapsto \gamma^{1-\alpha}$ applied to γ_1, γ_2, and γ, we get that the dynamic energy decreases. So, the total energy consumption decreases.

All processors finish at the same time. Let us assume that the processors do not finish at the same time. Then, there is a processor finishing before at least one other processor. We apply a factor $c < 1$ to all the speeds on this processor such that the $C_{\max}$ does not increase. The static energy does not change, but the dynamic energy decreases. So the total energy consumption decreases. □

Theorem 12. *If an algorithm without speeds outputs a schedule with* $\max(W_i - W_j) = \delta$*, then we can transform it in polynomial time, with the optimal choice of speeds, into a schedule with* $E \leq (1 + \frac{m\delta}{W})\text{OPT}$*, where* OPT *is the minimal energy consumption that could be attained.*

Proof. Using Lemma 12, if the assignment of tasks to processors is fixed, then we only have to choose a constant σ such that processor j runs at speed $\frac{\sigma \times W_j}{W_{\max}}$. The energy we get for a given σ is then:

$$\begin{aligned} E_{W_1,\ldots,W_m}(\sigma) &= \frac{W_{\max}}{\sigma} \times m \times P_{stat} + \sum_j W_j \times \frac{W_{\max}}{\sigma \times W_j} \times \left(\frac{\sigma \times W_j}{W_{\max}}\right)^{\alpha} \\ &= \frac{1}{\sigma} \times W_{\max} \times m \times P_{stat} + \sigma^{\alpha-1} \times \sum_j \frac{W_j^{\alpha}}{W_{\max}^{\alpha-1}}. \end{aligned}$$

This energy consumption is minimized for $\sigma = \frac{W_{\max} \sqrt[\alpha]{m \times P_{stat}}}{\left\|\vec{W}\right\|_{\alpha} \sqrt[\alpha]{\alpha-1}}$.

Now, the minimal energy $E^{(min)}_{W_1,\ldots,W_m}$ for this task assignment is:

$$E^{(min)}_{W_1,\ldots,W_m} = (m \times P_{stat})^{\frac{\alpha-1}{\alpha}} \times \left((\alpha-1)^{\frac{1}{\alpha}} + (\alpha-1)^{\frac{1-\alpha}{\alpha}}\right) \times \left\|\vec{W}\right\|_{\alpha}.$$

We now prove that over all valid $\vec{W}$, this quantity is bounded by $E^{(min)}_{\frac{W}{m},\ldots,\frac{W}{m}}$. We do so through induction over m.

We have:

$$\left\|\overrightarrow{\frac{W}{m}, \frac{W}{m}, \ldots, \frac{W}{m}}\right\|_{\alpha} = \sqrt[\alpha]{m}\frac{W}{m}.$$

Let $f(W,m) = \min_{\vec{W} \in \mathbb{R}^m \mid \sum W_i = W} \left\|\vec{W}\right\|_{\alpha}$.

For $m = 1$, we have:

$$f(W, 1) = W = \sqrt[\alpha]{m}\frac{W}{m}.$$

Now, we assume that for $m \geq 1$, we have $f(W, m) = \sqrt[\alpha]{m}\frac{W}{m}$.
By definition of f,

$$f(W, m+1) = \min_{\overrightarrow{W} \in \mathbb{R}^{m+1} | \sum_{i=1}^{i \leq m+1} W_i = W} \left\| \overrightarrow{W} \right\|_\alpha.$$

Therefore, fixing W_{m+1} and by definition of the α-norm,

$$\begin{aligned} f(W, m+1) &= \min_{w \in [0,W]} \min_{\overrightarrow{W} \in \mathbb{R}^{m+1} | \sum_{i=1}^{i \leq m} W_i = W - w \text{ and } W_{m+1} = w} \left\| \overrightarrow{W} \right\|_\alpha \\ &= \min_{w \in [0,W]} \min_{\overrightarrow{W} \in \mathbb{R}^m | \sum W_i = W - w} \sqrt[\alpha]{w^\alpha + \sum W_i^\alpha} \\ &= \sqrt[\alpha]{\min_{w \in [0,W]} w^\alpha + \min_{\overrightarrow{W} \in \mathbb{R}^m | \sum W_i = W - w} \sum W_i^\alpha}. \end{aligned}$$

Now, by definition of f and using the induction hypotheses, we obtain:

$$\begin{aligned} f(W, m+1) &= \sqrt[\alpha]{\min_{w \in [0,W]} w^\alpha + f(W - w, m)^\alpha} \\ &= \sqrt[\alpha]{\min_{w \in [0,W]} w^\alpha + m\frac{(W-w)^\alpha}{m^\alpha}} \\ &= \frac{\sqrt[\alpha]{m}}{m} \sqrt[\alpha]{\min_{w \in [0,W]} m^{\alpha-1} w^\alpha + (W-w)^\alpha}. \end{aligned}$$

We now need to study the variations of $g(w) = m^{\alpha-1}w^\alpha + (W-w)^\alpha$ to find the minimum value of $f(W, m+1)$. We have:

$$g'(w) = \alpha \times m^{\alpha-1} \times w^{\alpha-1} - \alpha \times (W-w)^{\alpha-1},$$

so we have:

$$\begin{aligned} g'(w) < 0 &\Leftrightarrow \alpha \times m^{\alpha-1} \times w^{\alpha-1} - \alpha \times (W-w)^{\alpha-1} < 0 \\ &\Leftrightarrow \alpha \times m^{\alpha-1} \times w^{\alpha-1} < \alpha \times (W-w)^{\alpha-1}. \end{aligned}$$

Therefore, since $\alpha > 0$, $g'(w) < 0 \Leftrightarrow m^{\alpha-1} \times w^{\alpha-1} < (W-w)^{\alpha-1}$. Now, using that $x \mapsto x^{\alpha-1}$ is an increasing function,

$$g'(w) < 0 \Leftrightarrow m \times w < W - w \Leftrightarrow w < \frac{W}{m+1}.$$

Meaning that the minimum value for $g(w)$ is reached for $w = \frac{W}{m+1}$, so we have:

$$f(W, m+1) = \sqrt[\alpha]{m+1}\frac{W}{m+1}.$$

So we have that $E^{(min)}_{\frac{W}{m},\dots,\frac{W}{m}}$ is a lower bound of the energy consumption of a schedule, with:

$$E^{(min)}_{\frac{W}{m},\dots,\frac{W}{m}} = (m \times P_{stat})^{\frac{\alpha-1}{\alpha}} \times \left((\alpha-1)^{\frac{1}{\alpha}} + (\alpha-1)^{\frac{1-\alpha}{\alpha}}\right) \times \sqrt[\alpha]{m+1}\frac{W}{m+1}.$$

In particular, as there exists a schedule with energy consumption OPT, we have:

$$\mathrm{OPT} = E^{(min)}_{W_1^{\mathrm{OPT}},\dots,W_m^{\mathrm{OPT}}}$$

that is minored by $E^{(min)}_{\frac{W}{m},\dots,\frac{W}{m}}$, meaning that:

$$P_{stat}^{\frac{\alpha-1}{\alpha}} \times \left[(\alpha-1)^{\frac{1}{\alpha}} + (\alpha-1)^{\frac{1-\alpha}{\alpha}}\right] \times W \leq \mathrm{OPT}.$$

As for the worst case, we know that $\min W_j \leq \frac{W}{m}$. Let $\overrightarrow{W_{worst}}$ be the vector maximizing the α-norm under the constraint that $W_j - W_i \leq \delta$. For this vector, we have $\max W_j \leq \frac{W}{m} + \delta$, so we get:

$$\left\|\overrightarrow{W_{worst}}\right\| \leq \sqrt[\alpha]{m}\left(\frac{W}{m} + \delta\right).$$

Now, if an algorithm $\mathcal{A}$ produces a schedule with energy $E_{\mathcal{A}}$ such that $W_j - W_i \leq \delta$ for all i, j, then we have:

$$\begin{aligned}\frac{E_{\mathcal{A}}}{E_{\mathrm{OPT}}} &\leq \frac{\frac{W}{m} + \delta}{\frac{W}{m}} \\ &\leq \frac{W + m \times \delta}{W} \\ &\leq 1 + m \times \frac{\delta}{W},\end{aligned}$$

which concludes the proof. □

Proposition 2. *The energy consumption of* SLACK *differs from the optimal one by at most* $m^2\Delta_{\mathcal{X},n}\frac{\mathrm{OPT}}{W}$*:*

$$0 \leq \frac{E(n, m, \mathcal{X}) - \mathrm{OPT}}{\mathrm{OPT}} \leq \frac{m^2\Delta_{\mathcal{X},n}}{W}.$$

Proof. Theorem 11 shows that for $\delta = (m-1) \times \Delta_{\mathcal{X},n}$, we have $\max\limits_{1\leq i,j\leq n}(W_i - W_j) \leq \delta$. Now using Theorem 12 with this premise, we directly get the desired result. □

Analogously to Proposition 1, Proposition 12 provides asymptotic results on SLACK used for optimizing the energy consumption. Further results can be obtained both for the uniform distribution in Corollary 5 and for the exponential distribution in Corollary 6. Intuitively, the result shows that the relative difference between the energy provided by the adapted SLACK algorithm and the optimal energy consumption converges to 0 almost surely, when $n \to +\infty$, with a speed at least $\frac{m^2 \log n}{n^2}$ for the uniform distribution and $\frac{m^2 \log \log n}{n}$ for an exponential distribution.

It is proved in [**devroye1981uniform**] that, almost surely,

$$\limsup_{n\to+\infty} \left(\frac{n\Delta_{\mathcal{U}[0,1],n} - \ln n}{2 \log n} \right) = 1.$$

Corollary 5. *When using* SLACK *as a base scheduling algorithm with the speed strategy exposed in Lemma 12 with uniform distribution for the tasks, one has almost surely*

$$\limsup_{n\to+\infty} \left(\frac{E_{\text{SLACK}}(n, m, \mathcal{U}[0,1]) - \text{OPT}}{\text{OPT}} \times \frac{n^2}{2(2+\ln 2)m^2 \log n} \right) \leq 1.$$

Proof. First, we simply rewrite the result from [**devroye1981uniform**]:

$$\limsup_{n\to+\infty} \left(\frac{n\Delta_{\mathcal{U}[0,1],n} - \ln n}{2 \log n} \right) = 1 \tag{4.4}$$

$$\limsup_{n\to+\infty} \left(\frac{n\Delta_{\mathcal{U}[0,1],n}}{2 \log n} - \frac{\ln n}{2 \log n} \right) = 1 \tag{4.5}$$

$$\limsup_{n\to+\infty} \left(\frac{n\Delta_{\mathcal{U}[0,1],n}}{2 \log n} - \frac{\ln 2}{2} \right) = 1 \tag{4.6}$$

$$\limsup_{n\to+\infty} \left(\frac{n\Delta_{\mathcal{U}[0,1],n}}{2 \log n} \right) = 1 + \frac{\ln 2}{2} \tag{4.7}$$

$$\limsup_{n\to+\infty} \left(\frac{n\Delta_{\mathcal{U}[0,1],n}}{(2+\ln 2) \log n} \right) = 1 \tag{4.8}$$

Using Proposition 2, we have

$$\frac{E_{\text{SLACK}}(n, m, \mathcal{U}[0,1]) - \text{OPT}}{\text{OPT}} \leq \frac{m^2 \Delta_{\mathcal{U}[0,1],n}}{W} \tag{4.9}$$

$$\frac{E_{\text{SLACK}}(n, m, \mathcal{U}[0,1]) - \text{OPT}}{\text{OPT}} \times \frac{n^2}{2(2+\ln 2)m^2 \log n} \leq \frac{n}{2W} \times \frac{n\Delta_{\mathcal{U}[0,1],n}}{(2+\ln 2) \log n} \tag{4.10}$$

Now using Equation 4.8, and the fact that $\lim \frac{n}{2W} = 1$ almost surely with the law of large numbers, we get that almost surely

$$\limsup_{n\to+\infty} \left(\frac{n}{2W} \times \frac{n\Delta_{\mathcal{U}[0,1],n}}{(2+\ln 2) \log n} \right) = 1 \tag{4.11}$$

So using Equation (4.10), we get that almost surely

$$\limsup_{n\to+\infty}\left(\frac{E_{\text{SLACK}}(n,m,\mathcal{U}[0,1])-\text{OPT}}{\text{OPT}}\times\frac{n^2}{2(2+\ln 2)m^2\log n}\right)\leq 1. \tag{4.12}$$

□

It is proved in [**devroye1984exponential**] that if $\mathcal{E}_1$ is the exponential distribution of rate 1, then, almost surely,

$$\limsup_{n\to+\infty}\left(\frac{\Delta_{\mathcal{E}_1,n}}{\ln\ln n}\right)=1.$$

As the rate λ of an exponential distribution is a scaling parameter, we get that if $\mathcal{E}_\lambda$ is the exponential distribution of rate λ, then almost surely

$$\limsup_{n\to+\infty}\left(\frac{\lambda\Delta_{\mathcal{E}_\lambda,n}}{\ln\ln n}\right)=1.$$

Corollary 6. *When using* SLACK *as a base scheduling algorithm with the speed strategy exposed in Lemma 12 with exponential distribution of rate λ for the tasks, for any fixed $m\geq 2$, one has almost surely*

$$\limsup_{n\to+\infty}\left(\frac{E_{\text{SLACK}}(n,m,\mathcal{E}_\lambda)-\text{OPT}}{\text{OPT}}\times\frac{n}{m^2\ln\ln n}\right)\leq 1.$$

The proof is very similar to the one of Corollary 5.

4.6 Simulations

We first present the simulation setting in Section 4.6.1, before studying the δ_j's and β_j's in Section 4.6.2, and the energy consumption in Section 4.6.3.

4.6.1 Experimental setting

All the following experiments have been conducted on Python 3.8.10. Two types of instances have been used. Both instances have in common that the platform is composed of $m = 100$ processors.

Theoretical instances have been generated using the *random* package. These instances have been generated following commonly used random distributions: the uniform distribution, $\mathcal{U}[0,1]$; the exponential distribution of rate 1, $\mathcal{E}_1$; the distribution of cumulative distribution function $F(x) = x^\alpha$ where $0 < \alpha < \infty$ [**frenk1986**]. These simple distributions correspond to the ones for which there exist convergence results in the literature and they cover a wide range of situations.

Realistic instances have been generated using the experimental cumulative distribution functions of actual workloads [**feitelson2014**]. These real workloads can be found on the Parallel Workload Archive from the website https://www.cs.huji.ac.il/labs/parallel/workload/. We used three specific instances: KIT ForHLR II with 114,355 tasks; NASA Ames iPSC/860 with 18066 tasks; and San Diego Supercomputer Center (SDSC) DataStar with 84907 tasks.

4.6.2 Simulations: Study of δ_j and β_j

In this section, we describe the results of our simulations comparing the values of δ_j and β_j (the largest differences between the execution times of the processors and the tasks, as defined in Section 4.3) over the execution of SLACK and LPT.

In Figures 4.1 and 4.2, we can see the evolution of the quantities studied in Section 4.4 when bounding the performance of SLACK. The quantities are:

- β_j the difference between the largest and the shortest task of pack j (i.e., at step j of the algorithm), it describes the imbalance between consecutive tasks during the execution of the algorithms;
- δ_j the difference between the largest processor and the shortest processor after step j of the algorithm (i.e., after allocating $j \times m$ tasks), it describes the imbalance between processors during the execution of the algorithms.

With these experiments, we can both investigate the relation we stated in Section 4.4, and investigate the unexplained "wave pattern" presented in [**benoit2021**].

In the case of tasks drawn through a uniform distribution with Figure 4.1, we observe that δ_j, the imbalance between processors, alternates between high and low values, in a sort of wave pattern. With a new representation of the pattern, we now present more elements explaining it. This pattern can be explained by the fact that the imbalance created by m consecutive tasks is then canceled by the m following tasks, as they have similar relative differences. Once the imbalance on the processors have decreased, the next m tasks will restore a new but smaller imbalance.

For most other distributions, on Figure 4.2, SLACK and LPT perform similarly in terms of makespan, which is characterized by the last value $\delta_{\frac{n}{m}}$. Out of our six examples, the only distribution for which SLACK performs significantly better than LPT is the distribution with cumulative distribution function $F(x) = x^{10}$, namely the one for which there are a few small tasks but many large ones.

A closer look at the evolution of δ_j and β_j gives more insights about the differences of execution between SLACK and LPT, and allows us to understand why SLACK performs better than LPT in some cases. Generally speaking, SLACK balances the different processors more quickly than LPT, and then keeps them balanced. In the specific case of $F(x) = x^{10}$, LPT performs significantly worse than SLACK because there is a high density of big tasks, and a low density of small tasks. It means that the big tasks are easy to balance whereas the small tasks are very different from each other. LPT finishes

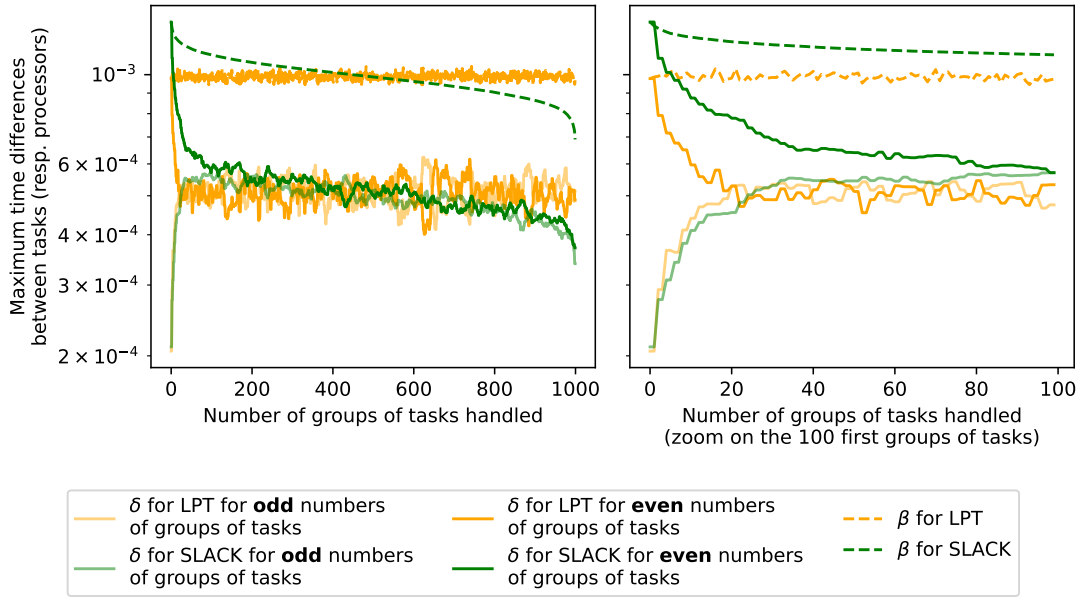


Figure 4.1: Evolution of δ_j and β_j (as defined in Section 4.3) during the execution of SLACK and LPT with the uniform distribution $\mathcal{U}[0,1]$ for the tasks. Each execution is done with $m = 100$ processors and $n = 100\,000$ tasks. The right graph is a zoomed version of the 100 first values of δ_j and β_j. Each point represents the average value of δ_j (resp. β_j) over 30 executions.

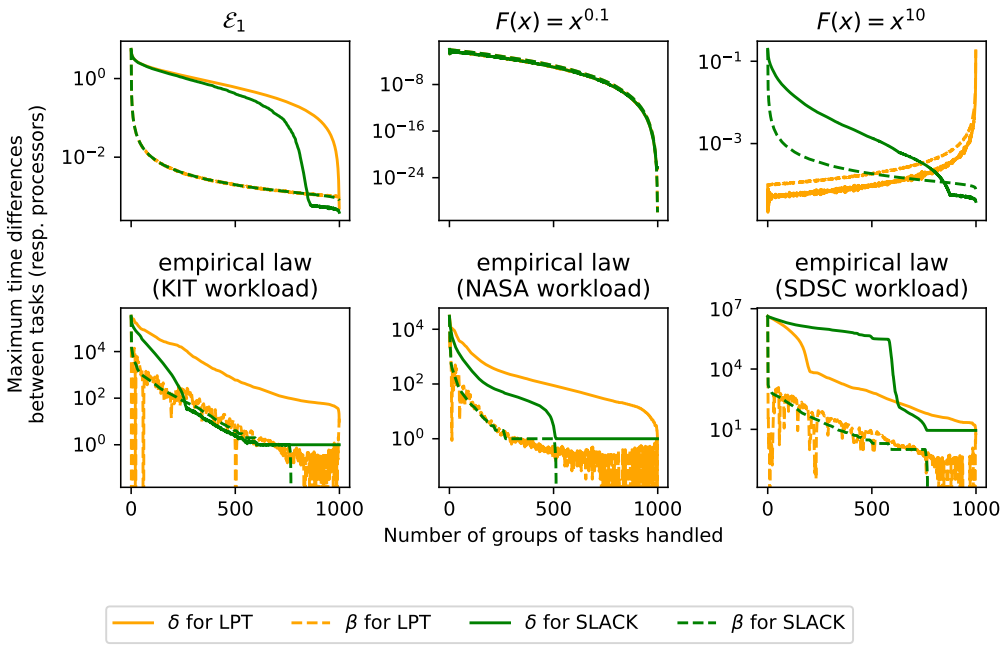


Figure 4.2: Evolution of δ_j and β_j (as defined in Section 4.3) during the execution of SLACK and LPT with various probability distributions for the tasks. Each execution is done with $m = 100$ processors and $n = 100\,000$ tasks. Each point represents the average value of δ_j (resp. β_j) over 30 executions.

its execution with small tasks that have a very high difference β_j, whereas SLACK is able to balance the processors using big tasks.

4.6.3 Simulations: Energy minimization

In this section, we describe the results of the simulations, evaluating the energy consumed by the schedules of the algorithms derived from LPT and SLACK (as defined in Section 4.3).

We do not directly consider the energy E found by an algorithm because the value of W can vary a lot depending on the instance. Instead, we consider the relative difference between the energy found by the algorithm and a lower bound on OPT, i.e., $\frac{E-E_{\frac{W}{m},\dots,\frac{W}{m}}}{E_{\frac{W}{m},\dots,\frac{W}{m}}}$. We have shown in the proof of Theorem 12 that $E_{\frac{W}{m},\dots,\frac{W}{m}}$ was indeed a lower bound on OPT.

The main conclusion that we can get from Figures 4.3 and 4.4 is that LPT and SLACK both perform very well on all created instances, both theoretical and realistic. The schedule that the two algorithms output is at most a few percents away from the optimal for very small instances, and the room for improvement rapidly decreases to less than $10^{-8}\%$ for larger instances.

It can be noted that SLACK performs better than LPT on average, even if they are both near optimal.

4.7 Conclusion

The optimization of parallel computing platforms, especially for energy purposes, is a challenging societal issue. In this chapter, we focus on SLACK, a recent heuristic that proves to be very efficient in practice for scheduling independent tasks on homogeneous machines. We have given, to the best of our knowledge, the first asymptotic performance results on the makespan for SLACK for tasks distributed randomly either uniformly or according to an exponential distribution. We have also shown how to adapt the numerous algorithms aiming at optimizing the makespan into algorithms dedicated to the optimization of the energy consumption. Based on SLACK, we were able to derive asymptotic energy performance results for both uniformly distributed tasks and exponentially distributed tasks. All these results exploit a common bound based of the shift between the most loaded and the least loaded processor during the execution of the heuristics. The experimental part of the chapter proposes an empirical comparison of these shifts for SLACK and LPT. Finally, the performance for the energy problem is also studied experimentally, and both SLACK and LPT are shown to be near optimal in terms of energy consumption. In the future, it would be interesting to explore other energy cost models. It would also be interesting to study the bi-objective problem that considers both the makespan and the energy consumption. Finally, monitoring the energy consumption of schedules on actual machines using the proposed algorithms should be explored.

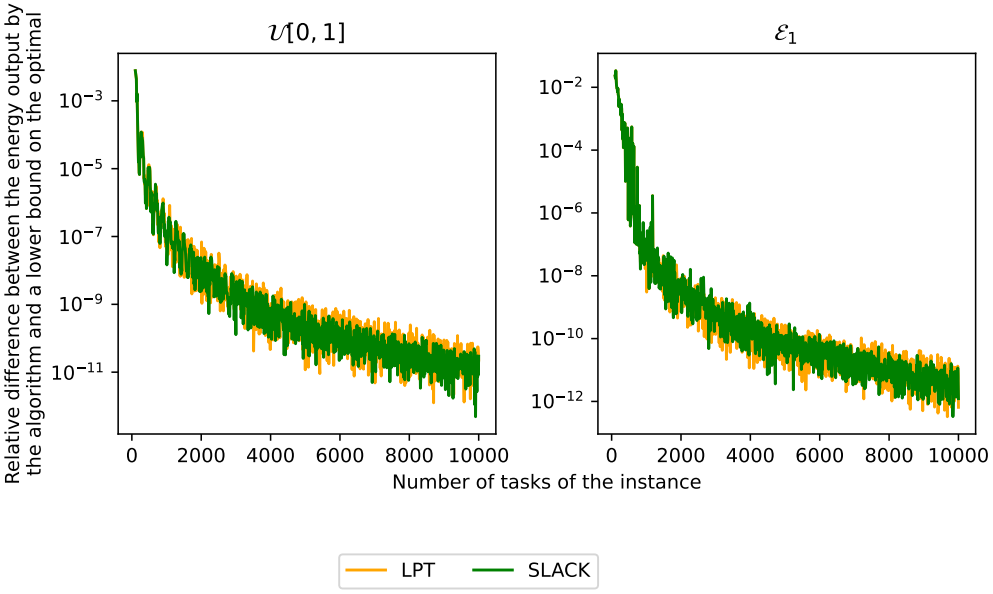


Figure 4.3: Relative difference between the energy found by SLACK or LPT with the speed strategy described in Theorem 12 and a lower bound on OPT, with various theoretical probability distributions for the tasks. Each execution is done with $m = 100$ processors. Each point represents the average value of energy over 30 executions.

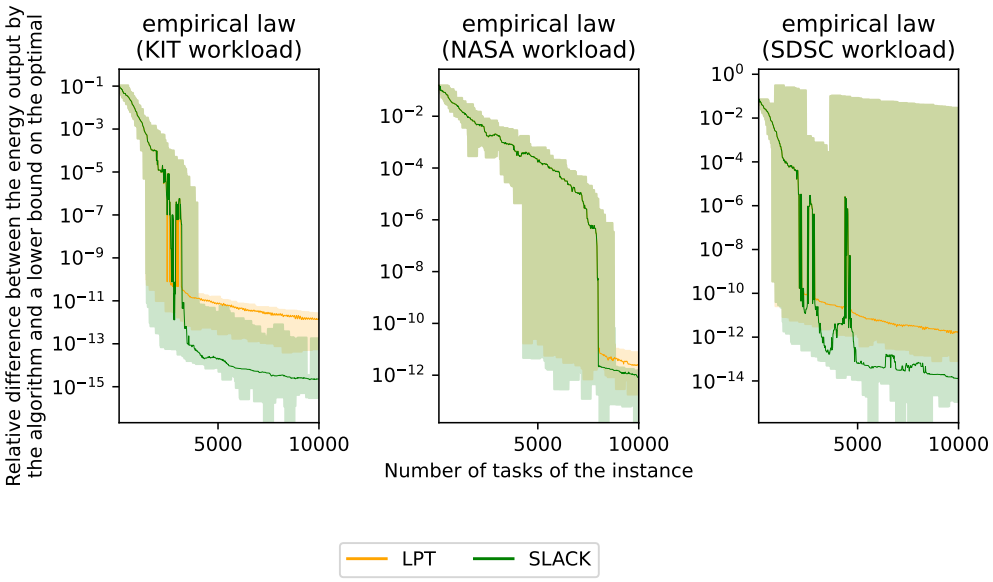


Figure 4.4: Relative difference between the energy found by SLACK or LPT with the speed strategy described in Theorem 12 and a lower bound on OPT, with various empirical probability distributions for the tasks. For each number of tasks, the execution is repeated 30 times with $m = 100$ processors. The thick lines represent the moving median, while the ribbons extend to the moving minimum and maximum over 45 values.

Chapter 5

Conclusion

Summary of results

In this thesis, we have tackled several task scheduling problems. Each of these problems considered either a number of tasks growing to infinity, the energy consumption of the schedule, or both. When relevant, we modeled the problem, proved its hardness through NP-completeness reductions, and proposed algorithms solving the problem. We have proved theoretical bounds and results for some already existing algorithms and for the proposed algorithms when there was no such result in the literature. For each problem, we have run and presented extensive simulations. These simulations allowed us to compare the theoretical bounds found to the actual performance of the algorithms. They also allowed us to compare the different algorithms proposed for a problem.

Chapter 2, published at Euro-Par 2021 [**benoit2021**], studies the makespan minimization problem with sequential tasks, when the amount of tasks grows to infinity, and the execution times of the tasks are sampled according to a specific distribution. We focused the study on two list-scheduling algorithms: LPT and SLACK. We first did a summary of existing results related to the asymptotic results of LPT. When the execution times of the tasks follow a specific probability distribution, we derived probabilistic results about the relative or absolute error of LPT when compared to OPT, depending on the specific probability distribution. We then derived new asymptotic results for the case when the execution times of the tasks follow the probability distribution, called the uniform integer composition. This distribution is different from usual distributions as the execution times of the tasks are not independent random variables, but still the almost sure optimality of LPT and SLACK has been proven. We also simulated several algorithms with various probability distributions, assessing the theoretical results that already exist in the literature.

Chapter 3, published at SBAC-PAD 2021 [**benoit2021shelf**] and extended in the Journal of Parallel and Distributed Computing [**benoit2023list**], focuses on the energy consumption minimization problem with moldable tasks. We modeled this optimization problem, providing its theoretical foundations. We studied two classes of schedules: list schedules and shelf schedules. In both cases, we designed several algorithms. We proved

that these algorithms are approximation algorithms with approximation between 2.08 and 3. For the optimization of a single shelf, we proved that it was tractable by providing an exact polynomial-time algorithm. Finally, we conducted extensive simulations to assess the quality of the provided algorithms.

Chapter 4, to be published at Euro-Par 2023 [**benoit2023asymptotic**], considers the makespan or energy consumption minimization problem with sequential tasks, when the amount of tasks grows to infinity, and the execution times of the tasks are sampled according to a specific distribution. We focused our study on algorithms based on SLACK and LPT. We showed how to adapt these algorithms to the energy minimization problems. We derived theoretical bounds for SLACK in the cases of both the makespan and the energy consumption. The proof for these bounds rely on a fundamental bound on the result of SLACK that is also proven in this chapter. We finally assessed these theoretical results through extensive simulations.

Perspectives

Work extending the subjects treated in this thesis include perspectives related to the first axis, i.e., to asymptotic stochastic considerations of scheduling problems and their algorithms. New distributions could be considered for the execution times of the tasks. For example, distributions that do not produce independent execution times could be studied, similarly to the uniform integer composition. Distributions with a null density near zero, could also be considered, as many existing results rely on the fact that the distributions are real and produce execution times arbitrarily close to zero. For example, the uniform distribution between a and b (e.g., $\mathcal{U}(a, b)$) could be studied more extensively. Another limitation of this thesis and, to the best of our knowledge, of existing work is that only the number of tasks n grows to infinity, while the number of processors m is considered to be constant. It would be interesting but harder to consider a growing number of processors, as the number of cores used by a recent supercomputer can grow large. It would also probably be possible to extend existing results related to the algorithm LPT to the newer algorithm SLACK.

There are also perspectives related to the second axis of the thesis, i.e., to the minimization of the energy consumption of a schedule. This thesis used a common energy model that separates the energy consumption into a static component and a dynamic component. However, other models exist, and it would be interesting to see which results can be derived with stronger or different models. We can cite two examples such as the polynomial model, where the energy consumption is assumed to be a polynomial function of the speed, and the convex function model, where the energy consumption is only assumed to be a convex function of the speed. It is also worth noting that the processors used in the simulations of this thesis are rather old. In order to do simulations with more recent processors, it would be necessary to take measurements through experiments on more recent HPC platforms. Finally, the problems studied here only consider one objective: either the makespan or the energy consumption. The reason is that the static part of the energy consumption already ensures that the makespan does not grow too

large. With a different energy model, it could be interesting to tackle the bi-objective problem, where both the makespan and the energy consumption are considered.

Finally, there are perspectives related to both axis. Every extension mentioned in the two previous paragraphs can also be applied to the situation where both axis are considered, as in Chapter 4. In both cases, we started with a rather simple model, but more complicated models exist in the literature. For instance, it is possible to add dependencies between tasks, in the form of a dependency graph describing which tasks must be finished before starting a task. We could consider some forms of uncertainty with execution times that are not perfectly known, or machines that might fail, requiring resilient algorithms. Heterogeneous platforms could also be considered, with machines that do not all behave in the same way. It would also be interesting to conduct experiments on actual HPC machines, as the systematic experimentation process of this thesis has been done through simulations. ‘

école doctorale **sciences pour l'ingénieur et microtechniques** SPIM

Titre : Bornes théoriques de problèmes d'ordonnancement et leurs applications à l'analyse asymptotique et la minimisation de la consommation d'énergie

Mots clés : Ordonnancement, Algorithmes, Calcul haute performance, Probabilité, Consommation d'énergie

Résumé : Les problèmes d'ordonnancement consistent à étudier comment affecter de façon automatique un ensemble de tâches à un ensemble de ressources en optimisant un ou plusieurs critères, comme par exemple le temps d'exécution total. En fonction des contraintes que l'on peut imposer sur les tâches, les ressources ainsi que sur les critères d'optimisation, il existe des dizaines de problèmes d'ordonnancement différents. Ces problèmes sont le plus souvent NP-complets. On ne sait donc pas trouver une solution garantie comme optimale en un temps raisonnable dans l'état actuel des connaissances humaines. Les approches pratiques s'appuient alors sur des heuristiques, qui consistent à trouver une solution rapidement dont le critère optimisé n'est pas nécessairement l'optimal mais qui en reste proche.

Dans cette thèse nous nous intéressons aux cas de tâches indépendantes et de ressources (processeurs) identiques avec deux critères d'optimisation : le temps d'exécution total pour finir toutes les tâches et la consommation énergétique totale, dans l'hypothèse où les processeurs peuvent avoir des vitesses variables et contrôlables. Dans ce cadre, nous étudions des heuristiques existantes et nous en proposons de nouvelles avec deux angles d'analyse théorique. D'une part nous fournissons des bornes asymptotiques probabilistes de convergences, dépendant de la distribution des tâches. Ces bornes permettent de garantir, lorsqu'il y a un grand nombre de tâches à réaliser, qu'avec une forte probabilité les heuristiques fournissent des solutions presque optimale. Ce type de résultat permet d'expliquer théoriquement l'excellent comportement dans les cas pratiques de l'heuristique LPT. D'autre part, nous fournissons différents ratio d'approximation, notamment dans le cas de l'énergie, prouvant que des algorithmes efficaces ne dévient pas trop d'une solution optimale qu'on ne sait pas calculer en temps raisonnable. Ces résultats sont obtenus en particuliers pour des tâches moldables, c'est-à-dire pouvant être exécutées chacune sur un nombre quelconque de processeurs. Nous proposons aussi des résultats mixant les deux types d'approches pour l'heuristique SLACK.

Title : Theoretical bounds for scheduling problems and their application to asymptotic analysis and energy consumption minimization

Keywords : Scheduling, Algorithms, High Performance Computing, Probabilitie, Energy Consumption

Abstract : Scheduling problems consist in studying how to assign a set of tasks automatically to a set of resources by optimising some criteria, such as the total execution time. Depending on the constraints imposed on the tasks, resources and optimisation criteria, there are dozens of different scheduling problems. These problems are often NP-complete, so we cannot find an optimal solution in a reasonable time with our current level of knowledge. The practical approaches are based on heuristics, consisting in finding quickly a solution which is not necessarily optimal, but which remains close to the optimal.

In this thesis we are interested in the case of independent tasks and identical resources (processors) with two optimization criteria: the total execution time to finish all tasks and the and the total energy consumed to execute all tasks, under the assumption that the processors can have variable and controllable speeds. In this context, we study existing heuristics and propose new ones with two theoretical angles. On the one one hand we provide probabilistic asymptotic bounds for convergences, depending on the theoretical distribution of tasks. These bounds make it possible to guarantee, when there is a large number of tasks that, with a high probability, the heuristics provide near-optimal solutions. This type of result theoretically explain the excellent behaviour of the LPT heuristic. On the other hand, we provide different approximation ratios, notably in the case of the energy minimization, proving that efficient algorithms do not deviate too much from optimal solutions, which we do not know how to compute quickly enough. These results are obtained in particular for moldable tasks, i.e. tasks that can be executed on any number of processors. We also propose results combining the two types of approach for the SLACK heuristic.