# On the Computational Complexity of Periodic Scheduling

THÈSE N$^{O}$ 4513 (2009)

PRÉSENTÉE LE 16 OCTOBRE 2009

À LA FACULTÉ SCIENCES DE BASE
CHAIRE D'OPTIMISATION DISCRÈTE
PROGRAMME DOCTORAL EN MATHÉMATIQUES

ÉCOLE POLYTECHNIQUE FÉDERALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Thomas ROTHVOSS

acceptée sur proposition du jury:

Prof. K. Hess Bellwald, présidente du jury
Prof. F. Eisenbrand, directeur de thèse
Prof. S. Baruah, rapporteur
Prof. M. A. Shokrollahi, rapporteur
Prof. M. Skutella, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2009

# Abstract

In *periodic scheduling* jobs arrive regularly and have to be executed on one or several machines or processors. An enormous amount of literature has been published on this topic, e.g. the founding work of Liu & Layland [LL73] is among the top cited computer science papers, according to Citeseer database. The majority of these papers is mainly concerned about practical applications and several important theoretical questions remained unsolved.

Let $\mathcal{S}$ be a set of *periodic tasks*, where each task $\tau \in \mathcal{S}$ releases a job of *running time* $c(\tau)$ and *relative deadline* $d(\tau)$ at each integer multiple of its *period* $p(\tau)$. In the single-processor scheduling one considers rules like *Earliest-Deadline-First* (EDF) or *Rate-monotonic* (RM) scheduling, that determine the order, in which the jobs have to be processed. The principal question here is to predict, whether such a schedule is feasible, i.e. whether all jobs are finished before their individual deadlines.

It was well-known that to validate the feasibility of an EDF-schedule, one has to evaluate the condition

$$\forall t \geq 0 : \sum_{\tau \in \mathcal{S}} \max\left\{ \left( \left\lfloor \frac{t - d(\tau)}{p(\tau)} \right\rfloor + 1 \right), 0 \right\} \cdot c(\tau) \leq t.$$

But the complexity status of this test remained unknown, despite of many heuristic approaches in the literature. We prove that testing this condition is **coNP**-hard even in special cases, answering an open question of Baruah & Pruhs [BP09].

For a *static-priority* schedule of *implicit-deadline* tasks, i.e. $d(\tau) = p(\tau)$, it is known that the jobs of a task $\tau$ will meet their deadlines if and only if there exists a fix-point $r \in [0, p(\tau)]$ of the equation

$$c(\tau) + \sum_{\tau' \prec \tau} \left\lceil \frac{r}{p(\tau')} \right\rceil \cdot c(\tau') = r,$$

where the sum ranges over all tasks with priority higher than $\tau$. We settle the complexity status of this problem by proving its **NP**-hardness, even if one asks for modest approximations.

Both results are achieved by bridging the more practically oriented area of *Real-time scheduling* and the field of *algorithmic number theory.* In fact, the intractability follows by a chain of reductions to *simultaneous Diophantine approximation (SDA)*, which is a classical problem in the *geometry of numbers* and deals with finding a small denominator $Q \in \{1, \ldots, N\}$, that yields a good approximation to a given vector of rational numbers $\alpha \in \mathbb{Q}^n$, i.e. $\max_{i=1,\ldots,n} |Q\alpha_i - \lceil Q\alpha_i \rfloor| \leq \varepsilon$. We strengthen existing hardness results for SDA such that they admit intractability results also for a directed version, where the goal is to find a $Q \in \{1, \ldots, N\}$ with $\max_{i=1,\ldots,n}(\lceil Q\alpha_i \rceil - Q\alpha_i) \leq \varepsilon$. We show that this problem remains **NP**-hard if one asks for an approximation that may exceed the bound $N$ by a factor of $n^{\mathcal{O}(1/\log\log n)}$ and the error bound $\varepsilon$ by $2^{n^{\mathcal{O}(1)}}$. As an application we are able to answer an open question of Conforti et al. [CSW08] negatively by obtaining **NP**-hardness for the problem of optimizing a linear function over a mixing set $\{(s, y) \in \mathbb{R}_{\geq 0} \times \mathbb{Z}^n \mid s + a_i\, y_i \geq b_i\ \forall i = 1, \ldots, n\}$. Furthermore we obtain that, regardless to the used $\| \cdot \|_p$-norm, the problem of finding a shortest *positive* vector in a lattice is not approximable within a factor of $n^{\mathcal{O}(1/\log\log n)}$, unless $\mathbf{NP} = \mathbf{P}$.

For scheduling *constrained-deadline* tasks (i.e. $d(\tau) \leq p(\tau)$) with a static-priority policy one possibly needs machines, which are a factor $f > 1$ faster than it is needed for the EDF policy. Baruah & Burns [BB08] sandwiched this factor between 1.5 and 2. We pinpoint this value to $f = 1/\Omega \approx 1.76$ by characterizing the properties of task systems, which attain this value. Here, $\Omega \approx 0.56$ is a well known constant from complex calculus, defined as real root of $x \cdot e^x = 1$.

We further consider a *multiprocessor* setting, where a given set of tasks has to be assigned to machines, such that each partition is RM-schedulable and the number of occupied processors is minimized. For the popular case that the tasks are equipped with *implicit-deadlines*, i.e. $d(\tau) = p(\tau)$, we develop new algorithms as well as intractability results:

- We state an asymptotic PTAS under resource augmentation. This is achieved by two concepts: First we derive that using a merging and clustering procedure, the instance can be modified, such that the utilization of tasks is lowerbounded by a constant, without excluding near-optimal solutions. Secondly, we introduce a relaxed notion of feasibility, which yields a drastic reduction of complexity and admits to compute solutions via dynamic programming.

- We introduce a simple First-Fit-like heuristic and obtain that this algorithm behaves nearly optimal on average, by proving that the expected waste is upperbounded by $\mathcal{O}(n^{3/4}(\log n)^{3/8})$, if the instance of $n$ tasks is generated randomly. Here, we assume that the utilization values $\frac{c(\tau)}{p(\tau)}$ of the tasks are drawn uniformly at random from $[0, 1]$.

- We state a new approximation algorithm with a running time of $\mathcal{O}(n^3)$, which can be sketched as follows: Create a graph with the tasks being the nodes and insert edges, if both incident tasks can be scheduled together on a processor. Then define suitable vertex costs and compute a min-cost matching, where the costs are the sum of edge costs plus the vertex costs of not covered nodes. From such a matching solution we can then extract a feasible multiprocessor schedule. The approximation ratio of this algorithm can be proven to tend to 3/2, improving over the previously best known ratio of 7/4 [BLOS95].

- We consider a column-based linear programming relaxation for this multiprocessor scheduling problem and derive that the asymptotic integrality gap is located between $4/3 \approx 1.33$ and $1 + \ln(2) \approx 1.69$.

- We disprove the existence of an asymptotic FPTAS, which yields that the problem is strictly harder to approximate than its special case of *Bin Packing*.

**Keywords:** Scheduling, combinatorial optimization, geometry of numbers, approximation algorithms, complexity theory, inapproximability, periodic tasks, simultaneous Diophantine approximation

# Zusammenfassung

Im *periodischen Scheduling* müssen regelmässig auftretende *Jobs* auf einer oder mehreren Maschinen bzw. Prozessoren abgearbeitet werden. In der Literatur wurde diesem Thema ein enormer Aufwand gewidmet, beispielsweise ist die grundlegende Arbeit von Liu & Layland [LL73] eine der am meisten zitierten Informatik-Veröffentlichungen überhaupt (siehe Citeseer-Datenbank). Die Mehrzahl dieser Arbeiten ist eher praktisch orientiert und einige fundamentale theoretische Fragestellungen blieben unbeantwortet.

Sei $\mathcal{S}$ eine Menge von *periodischen Tasks*, wobei ein Task $\tau \in \mathcal{S}$ zu jedem Vielfachen seiner *Periode* $p(\tau)$ einen Job mit *Ausführungszeit* $c(\tau)$ und *relativer Deadline* $d(\tau)$ generiert. Beim Single-Processor Scheduling betrachtet man Regeln wie *Earliest-Deadline-First* (EDF) oder *Rate-monotonic* (RM) Scheduling, welche die Reihenfolge angeben, in der die Jobs ausgeführt werden sollen. Die wesentliche Problemstellung ist es vorherzusagen, ob alle Jobs vor ihrer individuellen Deadline abgearbeitet werden.

Es ist seid langem bekannt, dass ein EDF-Schedule zulässig ist, genau dann wenn

$$\forall t \geq 0 : \sum_{\tau \in \mathcal{S}} \max \left\{ \left( \left\lfloor \frac{t - d(\tau)}{p(\tau)} \right\rfloor + 1 \right), 0 \right\} \cdot c(\tau) \leq t.$$

Trotz einer Vielzahl von heuristischen Ansätze in der Literatur, die das Testen dieser Bedingung untersuchen, blieb eine komplexitätstheoretische Einordnung bislang aus. Wir beweisen, dass das Testen der obigen Bedingung sogar in Spezialfällen **coNP**-schwierig ist, was eine offene Frage von Baruah & Pruhs [BP09] beantwortet.

In einem Schedule von Tasks mit *impliziten* Deadlines, d.h. $d(\tau) = p(\tau)$, mittels *statischer Prioritäten* werden genau dann alle Jobs eines Tasks $\tau$ ihre Deadline einhalten, wenn die Gleichung

$$c(\tau) + \sum_{\tau' \prec \tau} \left\lceil \frac{r}{p(\tau')} \right\rceil \cdot c(\tau') = r,$$

einen Fixpunkt $r \in [0, p(\tau)]$ aufweist, wobei über alle Tasks mit höherer Priorität summiert wird. Wir sind in der Lage zu zeigen, dass dieses Problem sogar im Bezug auf eine bescheidene Approximierbarkeit **NP**-schwierig ist.

Beide Resultate werden erreicht, indem wir eine Brücke zwischen dem eher praktisch orientieren Bereich des *Echtzeit-Scheduling* und dem Feld der *algorithmischen Zahlentheorie* schlagen. Die Herleitung der komplexitätstheoretischen Schwierigkeit erfolgt durch eine Kette von Reduktionen auf die *simultane Diophantische Approximation*, welche ein klassisches Problem der *Geometrie der Zahlen* darstellt und sich damit befasst, einen Zähler $Q \in \{1, \ldots, N\}$ zu finden, der einen gegebenen Vektor von rationalen Zahlen $\alpha \in \mathbb{Q}^n$ gut approximiert, d.h. $\max_{i=1,\ldots,n} |Q\alpha_i - \lceil Q\alpha_i \rfloor| \leq \varepsilon$. Wir verbessern bekannte Resultate für SDA derart, dass diese auch ein Nichtapproximierbarkeitsresultat für eine gerichtete Variante ermöglicht, in welcher es das Ziel ist ein $Q \in \{1, \ldots, N\}$ mit $\max_{i=1,\ldots,n}(\lceil Q\alpha_i \rceil - Q\alpha_i) \leq \varepsilon$ zu finden. Wir beweisen, dass das Problem **NP**-schwierig bleibt, sogar wenn die Schranke $N$ um einen Faktor von $n^{\mathcal{O}(1/\log\log n)}$ and die Fehlerschranke $\varepsilon$ um einen Faktor von $2^{n^{\mathcal{O}(1)}}$ überschritten werden darf. Als Nebenprodukt können wir eine offene Frage von Conforti et al. [CSW08] beantworten, in dem wir zeigen, dass das Optimieren einer linearen Zielfunktion über einer *Mixing Set* $\{(s, y) \in \mathbb{R}_{\geq 0} \times \mathbb{Z}^n \mid s + a_i\, y_i \geq b_i \; \forall i = 1, \ldots, n\}$ **NP**-schwierig ist. Eine weitere Anwendung besteht darin, dass unter der Annahme **NP** $\neq$ **P** das Problem einen *positiven* Vektor in einem Gitter zu finden unabhängig von der verwendeten $\|\cdot\|_p$-norm nicht innerhalb eines Faktors von $n^{\mathcal{O}(1/\log\log n)}$ in Polynomialzeit zu approximieren ist.

Damit das Schedule von *Constrained-Deadline* Tasks (d.h. $d(\tau) \leq p(\tau)$) mit statischen Prioritätszuweisungen zulässig ist, benötigt man Maschinen, die im ungünstigsten Fall einen Faktor $f > 1$ schneller sind, als dies für EDF-Scheduling notwendig wäre. Baruah & Burns [BB08] zeigen, dass dieser Faktor zwischen 1.5 and 2 liegt. Wir sind in der Lage, diese Konstante mit $f = 1/\Omega \approx 1.76$ exakt zu bestimmen, indem wir die Eigenschaften von Tasksystemen beschreiben, bei denen dieser Wert angenommen wird. Dabei ist $\Omega \approx 0.56$ eine aus der komplexen Analysis bekannte Konstante, welche als reelle Lösung von $x \cdot e^x = 1$ definiert ist.

Wir betrachten auch *Multi-Processor* Scheduling, bei dem eine gegebene Menge von Tasks auf Maschinen verteilt werden müssen, so dass das RM-Schedule von jedem Prozessor zulässig ist und die Anzahl der benötigten Prozessoren minimiert wird. Für den populären Fall, dass die Tasks *implizite Deadlines* haben, d.h. $d(\tau) = p(\tau)$, entwickeln wir neue Algorithmen sowie Komplexitätsresultate:

- Wir liefern ein asymptotisches PTAS unter *Resource Augmentation*. Dies erreichen wir durch zwei Schlüsselkonzepte: Durch das Vereinen von Tasks können wir die Eingabeinstanz so modifizieren, dass die Utilization jedes Tasks von unten durch eine Konstante beschränkt ist, ohne dass dies fast-optimale Lösungen ausschliesst. Dann relaxieren wir die Zulässigkeitsbedingung was eine drastische Reduktion des Rechenaufwands zur Folge hat und uns erlaubt, Lösungen mit dynamischer Programmierung zu berechnen.

- Wir führen einen einfachen First-Fit-basierten Algorithmus ein und beweisen, dass dieser Lösungen mit einem erwarteten *Waste* von höchstens $\mathcal{O}(n^{3/4}(\log n)^{3/8})$ liefert. Dabei nehmen wir an, dass die Utilization $\frac{c(\tau)}{p(\tau)}$ zufällig aus $[0, 1]$ gezogen wird.

- Wir präsentieren einen neuen Approximationalgorithmus mit einer Laufzeit von $\mathcal{O}(n^3)$, welchen man wie folgt skizzieren kann: Wir legen einen Graphen an, dessen Knoten die Tasks sind und fügen Kanten ein, wenn das RM-Schedule der inzidenten Tasks zulässig ist. Für geeignete Knoten- und Kantengewichte liefert ein Minimalkostenmatching in diesem Graphen eine Verteilung der Tasks, die einer asymptotischen 3/2-Approximation entspricht. Dies ist eine Verbesserung gegenüber der bislang besten Approximationsgarantie von 7/4 [BLOS95].

- Wir betrachten die spaltenbasierte LP-Relaxierung und zeigen, dass der asymptotische Integrality Gap dieser Formulierung zwischen $4/3 \approx 1.33$ und $1 + \ln(2) \approx 1.69$ liegt.

- Wir können zeigen, dass es unter Standardannahmen kein asymptotisches FPTAS für dieses Problem geben kann, also Multiprocessor Scheduling echt schwerer zu approximieren ist als sein Spezialfall *Bin Packing*.

**Schlüsselworte:** Scheduling, kombinatorische Optimierung, Geometrie der Zahlen, Approximationsalgorithmen, Komplexitätstheorie, Nichtapproximierbarkeit, periodische Tasks, simultane Diophantische Approximation

# Acknowledgements

First of all I want to thank my supervisor Fritz Eisenbrand for guiding me throughout this work, numerous discussions and suggestions.

I'm grateful to Sanjoy K. Baruah, for giving very helpful advices concerning the field of Real-time scheduling and to Martin Skutella and Amin Shokrollahi for being rapporteur, despite of lack of time.

Thanks to Gennady Shmonin for his free LaTeX support and to Andreas Karrenbauer for collaboration and intensive discussions, whether $n^{100}$ is the running time of an efficient algorithm. I won't forget Kai Gehrs for various discussions on student working ethic, politics and Dr. House. Of course, thanks to Dömötör, Genna, Martin, Nicolai, Andreas, Padmini, Rado, Jarek, Fabrizio and Laura for helping me to enjoy *la vie* and not letting me finish this thesis too early. I'm grateful to Jocelyne Blanc for guiding me around various bureaucratic obstacles.

I want to thank my family, especially my parents, for their support and my friends in Germany (in alphabetical order, without a guarantee of completeness) André, Basti, Christoph, Holger, Kathi, Majo, Malte and Theo for preserving our long-distance friendship.

Finally, I'm deeply convinced that I would not have been able to state the complexity theoretic parts of this thesis, without having attended the brilliant lecture *Komplexitätstheorie* by Ingo Wegener, who left us too early on the 26.11.2008. Farewell, Ingo.

# Contents

# Chapter 1

# Introduction

*"Cras legam*[1]*."*
—Gaius Iulius Caesar

*Combinatorial Optimization* deals with making an optimum choice among a finite number of possible combinations. To give an example, for moving we usually need to pack all our belongings into boxes, such that the boxes are not overpacked and the number of boxes is minimized. Typically the number of combinations is astronomically large, hence even the fastest existing computers would need billions of years to try out all possibilities. But many problems are well-structured and sophisticated problem-based algorithms can find optimal solutions in reasonable time, where in theoretical computer science one considers a running time as acceptable, if it is bounded by a polynomial in the input size. Other problems have been studied for decades, without admitting such an algorithm. On the other hand no one has been able to disprove the existence of efficient algorithms for the mentioned problems. The $\mathbf{NP} \neq \mathbf{P}$-*conjecture* says that, for the whole class of so called **NP**-*complete* problems, there is no polynomial time algorithm. Consequently researchers investigate *approximation algorithms*, which efficiently find solutions, whose quality is within a provably small distance, say 10%, from the optimum value. As we will see in this thesis, other problems forbid even modest approximations.

Among such combinatorial problems, we are studying *periodic scheduling*, where tasks $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$ are given, which release jobs of running time $c(\tau_i)$ and relative deadline $d(\tau_i)$ at each integer multiple of the period $p(\tau_i)$. We consider *preemptive scheduling policies*, i.e. rules that determine the order, in which the jobs are processed. For example the *Earliest-Deadline-First* policy executes at any time that job, whose deadline comes next. Other rules assign fixed priorities to each task: for example, the *Rate-monotonic* scheduling policy assigns higher priorities to tasks with smaller periods. If the jobs can be processed by some rule, such that they meet their deadlines, then it is known that also the EDF-policy produces a feasible order. In other words, EDF is an *optimal* scheduling policy. For task systems with *implicit deadlines*, i.e. $d(\tau) = p(\tau)$, the RM-policy yields optimal static-priorities in the sense that if the schedule is infeasible, then it will be infeasible for any assignment of fixed priorities.

The complexity status for deciding the feasibility of these schedules had remained an

[1] *"I will read it tomorrow!"*, announced by Gaius Iulius Caesar when a warning was passed to him while being on the way to the senate on the 15th of March 44 BC.

open problem since the initial studies decades ago. E.g., for the EDF-schedulability one has to evaluate the condition

$$\forall t \geq 0 : \sum_{\tau \in \mathcal{S}} \max \left\{ \left( \left\lfloor \frac{t - d(\tau)}{p(\tau)} \right\rfloor + 1 \right), 0 \right\} \cdot c(\tau) \leq t,$$

which involves a single variable $t$ and a floor operation. To obtain a hardness result, one has show that given an oracle for testing the above condition, one could also solve problems, for which efficient algorithms are unlikely to exist and consequently also the initial problem must be hard to solve. But the only problem that shares at least slight similarities and hence could serve as a starting point for a *reduction* is the so called *simultaneous Diophantine approximation* problem, which deals with finding nominators and a small *common* denominator such that the composed rational numbers do not differ much from a given set of reals. This problem, lying in the field of *algorithmic number theory* and the *geometry of numbers*, was already studied centuries ago by Dirichlet.

It can be considered as the central theme of this thesis, to apply results from the fields of theoretical computer science and the geometry of numbers to practically relevant problems in Real-time scheduling. We will see that as a kind of a by-product of this bridging, we can answer the question of the complexity status of the *mixing set problem* from the field of *integer programming.*

The second topic in this thesis is the *multiprocessor* setting, in which one asks to assign a given set of tasks to a minimal number of machines. The reader might observe that this goal bares some similarities with the above moving example, where the tasks correspond to our items and the machines mimic the boxes. Applying techniques from theoretical computer science and especially from the area of *approximation algorithms* will lead to several fruitful algorithmic and intractability results.

Clearly restricting the scheduling policies to static-priority assignments may cause that to achieve schedulability, processors are needed, which are a certain factor faster. Finally we are able to exactly quantify this factor for the class of constrained-deadline tasks, i.e. $d(\tau) \leq p(\tau)$.

Back to the cited epigraph, it seems that Caesar had assigned his priorities wrong — with fatal consequences.

## 1.1 Outline

Next, we briefly outline the content of the different chapters. Roughly spoken the first part (Chapter 2 to 6) of this thesis deals with algorithmic results, while the second part (Chapter 7 to 10) yields negative results, thus deals with inapproximability.

- Chapter 2: We begin with describing basic terms and concepts in scheduling, focusing on the theory of Real-time scheduling.

- Chapter 3: We obtain an asymptotic PTAS under resource augmentation for Rate-monotonic multiprocessor scheduling of implicit-deadline tasks. That means, for any fixed $\varepsilon > 0$, we can assign the tasks in polynomial time among $(1+\varepsilon)OPT + \mathcal{O}(1)$ many machines. Thereby the RM-schedule of the tasks on each processor is feasible, if the speed of each machine may be increased by a factor of $1+\varepsilon$. This is achieved by two concepts: First we derive that using a merging and clustering procedure, the instance can be modified such that the utilization of tasks is lowerbounded by a constant, without excluding near-optimal solutions. Secondly, we introduce a relaxed notion of feasibility, which yields a drastic reduction of complexity and admits to compute solutions via dynamic programming.

This chapter is a slight extension of the result of Eisenbrand & Rothvoß [ER08a].

- Chapter 4: We introduce a simple First-Fit-like algorithm for Rate-monotonic multiprocessor scheduling of implicit-deadline tasks. We show that the algorithm behaves nearly optimal on average, by proving that the expected waste is upperbounded by $\mathcal{O}(n^{3/4}(\log n)^{3/8})$ if the instance of $n$ tasks is generated randomly. Here, we assume that the utilization values $\frac{c(\tau)}{p(\tau)}$ of the tasks are drawn uniformly at random from $[0, 1]$. This result is achieved by proving that the algorithm behaves better than a specific restricted variant. The waste of this auxiliary algorithm can then be bounded by a reduction to well-known results from the average-case analysis for BinPacking. This reproduces Karrenbauer & Rothvoß [KR09].

- Chapter 5: We state a new approximation algorithm for Rate-monotonic multiprocessor scheduling of implicit-deadline tasks with a running time of $\mathcal{O}(n^3)$, which can be sketched as follows: Create a graph with the tasks being the nodes and insert edges if both incident tasks can be scheduled together on a processor. Then add suitable vertex costs and compute a min-cost matching. Here the cost is the sum of edge costs plus the vertex costs of not covered nodes. From such a matching solution we can then extract a feasible multiprocessor scheduling. The approximation ratio of this algorithm tends to 3/2, improving over the previously best known ratio of 7/4 [BLOS95].

- Chapter 6: We consider a column-based linear programming relaxation for Rate-monotonic multiprocessor scheduling of implicit-deadline tasks and derive that the asymptotic integrality gap is located between $4/3 \approx 1.33$ and $1 + \ln(2) \approx 1.69$. The upper bound is obtained by a randomized rounding scheme.

- Chapter 7: The *simultaneous Diophantine approximation (SDA)* problem deals with finding a small denominator $Q \in \{1, \ldots, N\}$, which yields a good approximation to a given set of rational numbers $\alpha \in \mathbb{Q}^n$, i.e. $\max_{i=1,\ldots,n} |Q\alpha_i - \lceil Q\alpha_i \rfloor| \leq \varepsilon$. We strengthen existing hardness results for SDA such that they admit intractability result also for the directed version, where the goal is to find a $Q \leq N$ with $\max_{i=1,\ldots,n}(\lceil Q\alpha_i \rceil - Q\alpha_i) \leq \varepsilon$. We obtain that this problem remains **NP**-hard if one asks for an approximation that may exceed the bound $N$ by a factor of $n^{\mathcal{O}(1/\log\log n)}$ and the error bound $\varepsilon$ by $2^{n^{\mathcal{O}(1)}}$. As an application we are able to answer an open question of Conforti et al. [CSW08] negatively by obtaining **NP**-hardness for the problem of optimizing a linear function over a mixing set $\{(s, y) \in \mathbb{R}_{\geq 0} \times \mathbb{Z}^n \mid s + a_i y_i \geq b_i\}$. As a by-product we obtain that, regardless to the used $\|\cdot\|_p$-norm, the problem of finding a shortest *positive* vector in a lattice is not approximable within a factor of $n^{\mathcal{O}(1/\log\log n)}$, unless $\mathbf{NP} = \mathbf{P}$.

  The chapter follows mainly Eisenbrand & Rothvoß [ER09].

- Chapter 8: In order to verify, that the Earliest-Deadline-First schedule of a set $\mathcal{S}$ of constrained-deadline tasks is feasible, one has to determine whether
$$\sum_{\tau\in\mathcal{S}}\left(\left\lfloor\frac{t-d(\tau)}{p(\tau)}\right\rfloor+1\right)\cdot c(\tau)\leq t$$
holds for all $t \geq 0$. Based on results from the previous chapter we derive that testing this condition is **coNP**-hard. This solves Problem 2 from the list of *Open Problems in Real-time Scheduling* [BP09].

  Secondly, to test, whether all jobs of an implicit-deadline task $\tau$ w.r.t. a static-priority schedule meet their deadline, one has to compute a minimum solution $r \geq 0$ of the equation
$$c(\tau)+\sum_{\tau'\prec\tau}\left\lceil\frac{r}{p(\tau')}\right\rceil c(\tau')=r,$$
where the sum ranges over the higher priority tasks. We obtain that this problem is **NP**-hard. In fact, even approximating the smallest $r$ up to a factor of $n^{\mathcal{O}(1/\log\log n)}$ is intractable. The complexity status of both problems had remained open for decades. The latter result is a strengthening of the paper of Eisenbrand & Rothvoß [ER08b].

- Chapter 9: By a reduction from 3-Partition, we can disprove the existence of an asymptotic FPTAS for multiprocessor scheduling of implicit-deadline tasks, unless $\mathbf{NP} = \mathbf{P}$. This yields the proof, that the problem is strictly harder than its special case of BinPacking. The chapter follows Eisenbrand & Rothvoß [ER08a].

- Chapter 10: For scheduling constrained-deadline tasks with a static-priority policy one possibly needs machines, which are a factor $f > 1$ faster than it is needed for the Earliest-Deadline-First policy. Baruah & Burns [BB08] sandwiched this factor between 1.5 and 2. We pinpoint this constant to $f = 1/\Omega \approx 1.76$ by characterizing the properties of task systems, which attain this value. Here $\Omega \approx 0.56$ is a well known constant from complex calculus. The content of this chapter is borrowed from Davis, Rothvoß, Baruah & Burns [DRBB09].

# Part I

# Algorithmic Results

# Chapter 2

# Preliminaries

## 2.1 The basics of scheduling

Suppose that a single processor (or machine) is available to us and we have jobs $J_1, \ldots, J_m$ that have to be executed on the machine, whereby at any time $t$ the machine can process at most one job, which is then called the *active* job w.r.t. $t$. Each job has a *running time*, a *release time* (a.k.a. *arrival time*) and a *deadline*, at which the job must be completed. In a *preemptive* schedule a job may be interrupted at any time $-$ and resumed later. Thereby the work, which has already been done is not lost, thus we do not have to process the full job again. In contrast, in non-preemptive scheduling once that a job is active, it cannot be interrupted before it is completed. As the name suggests, the *completion time* is that time, when the job is finished. The *response time* of a job is the difference of completion time and release time.

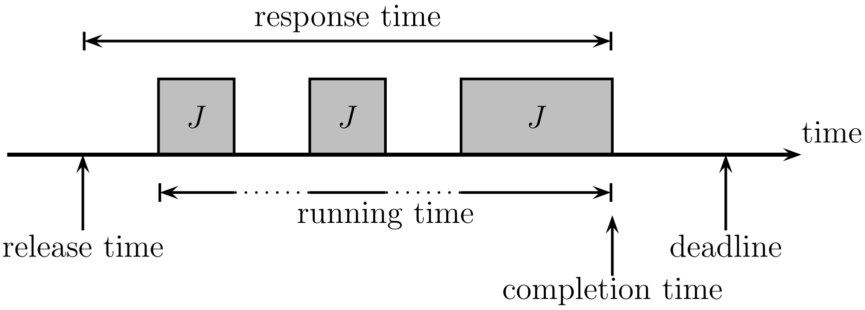


Figure 2.1: Processed job $J$ is preempted twice.

In general settings there might be precedence constraints, where for all jobs $J$ there is a (possibly empty) list of jobs, which have to be finished before $J$ may be processed. However we will not consider such constraints here.

A *task* $\tau$ is a finite or infinite sequence of jobs. Throughout this thesis we will consider the *periodic task model* in which each task $\tau$ releases a job with *execution time* $c(\tau)$ and *(relative) deadline* $d(\tau)$ every $p(\tau)$ time units, where $p(\tau)$ is termed the *period* of the task. We assume that the tasks are *synchronous*, i.e. there is a time, say 0, at which all tasks release a job simultaneously. In other words for each $z \in \mathbb{Z}_{\geq 0}$, a job of running time $c(\tau)$ and absolute deadline $z \cdot p(\tau) + d(\tau)$ is released at $z \cdot p(\tau)$.

In many settings, the deadlines are equal to the periods, i.e. we have *implicit deadlines* $d(\tau) = p(\tau)$ for all tasks $\tau$. If at least $d(\tau) \leq p(\tau)$ holds for all tasks, one speaks about *constrained deadlines*. The tasks have *arbitrary deadlines* (sometimes called *explicit deadlines*) if no such restriction holds.

A *schedule* (or *scheduling policy*) is an algorithm $A$ that decides, in which order the jobs are processed. If all jobs meet their deadlines in the schedule defined by $A$, then the jobs are called $A$-*schedulable*. If all task systems that are schedulable with some policy, are also $A$-schedulable, then $A$ is termed an *optimal* scheduling policy. In many applications the scheduler runs on a machine with few computation power like in a mobile or a car. Consequently one prefers simple scheduling algorithms, like *Earliest-Deadline-First* (EDF), where always that task is active, whose deadline comes next. Or we might assign priorities to the tasks so that at any time, we consider the queue of arrived but not yet finished jobs and choose that job, whose task has the highest priority, to be the active job. Since we want to compute the priorities in advance and are not willing to change them during time, we call this a *fixed-priority* (or *static-priority*) schedule. We should note that we will only consider preemptive scheduling, thus whenever a job of higher priority arrives, we would immediately stop a lower priority job in favour of the higher priority task.

EDF is an optimal dynamic priority scheduling policy for all considered periodic task model, see Dertouzos [Der74]. On the other hand if we are restricted to static priorities, for implicit-deadline tasks the *Rate-monotonic* (RM) scheduling policy is optimal (Liu & Layland [LL73]), in which the priority is higher, the smaller the periods are, i.e. the priority of $\tau$ is $1/p(\tau)$ (ties broken arbitrarily, but fixed). The *Deadline-monotonic* (DM) schedule, which assigns priority $1/d(\tau)$ to $\tau$ is optimal for constrained-deadline task systems [LW82]. Also in case of arbitrary deadlines, optimal priorities can be computed [Leh90].

The *response time* of a task $\tau$ (w.r.t. to a given schedule) is defined as the longest time, which ever elapses between arrival and accomplishment of a job of $\tau$. In other words it is the supremum of the response times of all its jobs. The job (or instance) of $\tau$, where the response time of the task is attained is called the *critical* job. Of course a schedule is feasible if and only if for each task, the response time does not exceed the deadline.

The following table gives an overview over optimal scheduling policies.

| **Setting** | | **optimal scheduler** | **complexity of feasibility test** |
|---|---|---|---|
| implicit-deadline | & static-priority | RM | ? |
| | & dynamic-priority | EDF | polynomial |
| constrained-deadline | & static-priority | DM | ? |
| | & dynamic-priority | EDF | **NP**-hard |
| arbitrary-deadline | & static-priority | [Leh90] | ? |
| | & dynamic-priority | EDF | **NP**-hard |

In the cases where the complexity status of the schedulability test is unknown, we are able to prove in Chapter 8 that computing the response time of a task is **NP**-hard, yielding a strong hint that the feasibility test itself is intractable.

In the *asynchrounous* setting, each task $\tau$ is additionally equipped with an offset $a(\tau)$, thus jobs are released at time $z \cdot p(\tau) + a(\tau)$ for all $z \in \mathbb{N}_0$, see [BG04] for an overview.

But we will not consider asynchronous tasks in this thesis.

In the *sporadic* task model neither release times nor running times are predetermined. There, $c(\tau)$ denotes the *worst-case execution time* and $p(\tau)$ denotes the *minimum inter-arrival time*. See Figure 2.2 for a visualization. But for the considered schedules, the worst-case is attained in a *synchronous arrival sequence*, that is when all tasks release jobs at time 0, all jobs fully use the worst-case execution time $c(\tau)$ and jobs arrive as early as permissible, see Baruah, Mok & Rosier [BMR90]. In other words, the sporadic task system is schedulable if and only if this is true for the corresponding periodic task system.

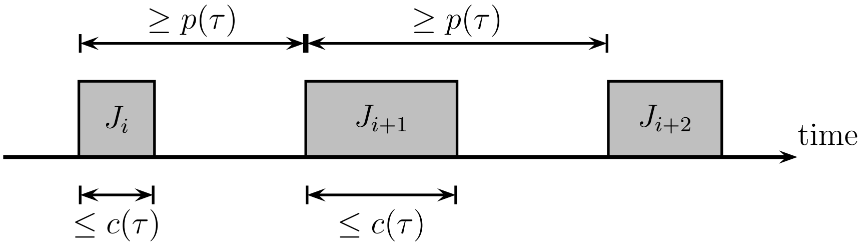


Figure 2.2: Jobs of a sporadic task $\tau$, scheduled alone on a processor.

If a task system $\mathcal{S}$ is scheduled on a processor with non-unit *speed* $\beta > 0$, this corresponds to a schedule of a modified task system $\mathcal{S}'$ on a unit-speed processor. Here $\mathcal{S}'$ emerges from $\mathcal{S}$ by dividing the running times of all tasks by $\beta$ or alternatively by multiplying all periods and deadlines by $\beta$.

For *multiprocessor* scheduling we have to distribute jobs (or tasks) among as few processors as possible, such that the jobs assigned to each processor are schedulable w.r.t. the given scheduling policy. For the sake of completeness we should mention that another popular variant is that the number of processors is fixed and one has to minimize the *makespan* (i.e. maximum completion time) or the (weighted) sum of completion times or response times. Such situations may be made even more complicated by different speed of processors, precedence constraints, enforcing or forbidding jobs to be processed on the same processor and so on and so forth. In view of the diversity of possible combinations and the fact that the *Handbook of Scheduling* [Leu04] has more than 1200 pages, the reader will understand, why we restrict ourselves to a specific scheduling problem.

## 2.2 RM-scheduling of implicit-deadline tasks

In this thesis we will mainly focus on Rate-monotonic scheduling of implicit-deadline tasks, which was introduced in 1973 by Liu & Layland [LL73]. Recall that we deal with tasks $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$, which we write abbreviated as $\tau_i = (c(\tau_i), p(\tau_i))$, since $d(\tau_i) = p(\tau_i)$. Each such task $\tau_i$ generates a job of running time $c(\tau_i)$ with release time $z \cdot p(\tau_i)$ and absolute deadline $(z+1) \cdot p(\tau_i)$ for all $z \in \mathbb{Z}_{\geq 0}$. If all jobs of task $\tau$ meet their deadlines in a RM-schedule, we say that $\tau$ is *feasible* w.r.t. $\mathcal{S}$. If all tasks are feasible, then $\mathcal{S}$ is called *RM-schedulable* or simply feasible. See Figure 2.3 for an example.

Since we already know the optimal priority assignments, the question, which naturally comes to our mind is, how to verify that the obtained schedule is indeed feasible; thus

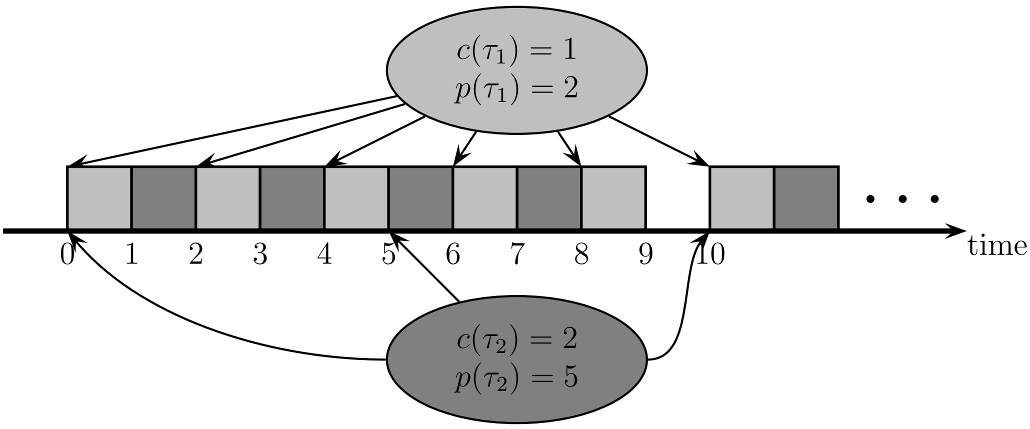


Figure 2.3: RM-schedule of an implicit-deadline task system $\tau_1 = (1, 2), \tau_2 = (2, 5)$. The arrows in the figure indicate the release times of the jobs. $\tau_1$ has higher priority than $\tau_2$. Consequently the first job of $\tau_2$ is interrupted by $\tau_1$ at time 2 and continues at time 3. Observe that $\operatorname{lcm}(p(\tau_1), p(\tau_2)) = 10$, thus the schedule repeats periodically and all jobs meet their deadlines.

the infinite number of jobs always meet their deadlines.

Let us define $r(\tau)$ to be the *response time* of task $\tau$, i.e. it is the longest time, which ever elapses between arrival and accomplishment of a job of $\tau$. Consider again Figure 2.3, then we have $r(\tau_1) = c(\tau_1) = 1$, since jobs of $\tau_1$ have the highest priority. On the other hand the first job of $\tau_2$ needs 4 time units and the second one 3, thus $r(\tau_2) = 4$. These values are very useful, since clearly the schedule is feasible if and only if $\forall \tau \in \mathcal{S} : r(\tau) \leq p(\tau)$. Consequently verifying feasibility may be reduced to finding response times. Let us denote $\tau_i \prec \tau_j$ if $\tau_i$ has higher priority than $\tau_j$. Thus $p(\tau_1) < \ldots < p(\tau_n)$ implies $\tau_1 \prec \ldots \prec \tau_n$.

Since in this setting it is always the *first* job, which defines the response time [LL73], $r(\tau)$ is the smallest fix-point of the *response time function* [LSD89]

$$D_{\tau,\mathcal{S}}(r) = c(\tau) + \sum_{\tau' \prec \tau} \left\lceil \frac{r}{p(\tau')} \right\rceil \cdot c(\tau'). \tag{2.1}$$

The reason for this is as follows: In the interval $[0, r(\tau)]$ one job of $\tau$ is executed, taking $c(\tau)$ time units and each task $\tau'$ with higher priority than $\tau$, is processed exactly $\lceil \frac{r(\tau)}{p(\tau')} \rceil$ many times in the interval, consuming $c(\tau')$ time units for each execution.

Applying this to the task system from Figure 2.3, we obtain $r(\tau_2) = 4$ as the smallest fix-point of

$$r(\tau_2) = c(\tau_2) + \left\lceil \frac{r(\tau_2)}{p(\tau_1)} \right\rceil \cdot c(\tau_1) = 2 + \left\lceil \frac{r(\tau_2)}{2} \right\rceil,$$

which matches the value, that we already came up with. To become more familiar with function $D_{\tau,\mathcal{S}}$, let us consider the former example of tasks $\tau_1 = (1, 2), \tau_2 = (2, 5)$ and add one more tasks $\tau_3 = (1/2, 12)$. We wonder, whether the RM-schedule will be still feasible. Since $\tau_3$ gets the lowest priority, all jobs of $\tau_1$ and $\tau_2$ will still meet their deadlines. To see, whether $\tau_3$ is feasible, let us inspect its response time function

$$D_{\tau_3,\mathcal{S}}(r) = \frac{1}{2} + \left\lceil \frac{r}{2} \right\rceil \cdot 1 + \left\lceil \frac{r}{5} \right\rceil \cdot 2,$$

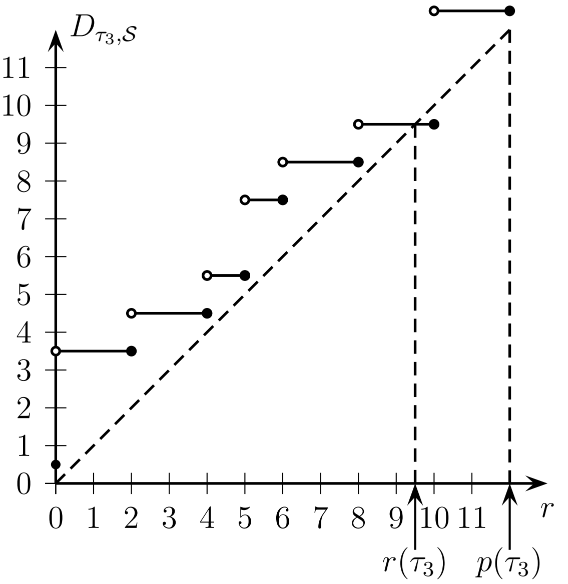


Figure 2.4: Function $D_{\tau_3,\mathcal{S}}(r) = \frac{1}{2} + \lceil \frac{r}{2} \rceil \cdot 1 + \lceil \frac{r}{5} \rceil \cdot 2$.

depicted in Figure 2.4. We observe that $r(\tau_3) = 9.5 \leq 12 = p(\tau_3)$, thus $\tau_3$ will meet the deadlines. Note that $D_{\tau,\mathcal{S}}(r)$ is monotonically increasing, thus from $D_{\tau,\mathcal{S}}(r) \leq r$, we can derive that there must be a fix point "before" $r$, formally $r(\tau) \leq r$. Furthermore a vector of response times $r(\tau)$ for $\tau \in \mathcal{S}$ can serve as a certificate for feasibility of the task system, since we can efficiently check whether indeed $D_{\tau,\mathcal{S}}(r(\tau)) \leq r(\tau)$. We do not have to verify that these values are the *smallest* fix-points.

But in Chapter 8 we will derive that even approximating response times is **NP**-hard. For $n = 2$ there is a simple exact schedulability criterion (cf. [Leu04]): The task set $\{\tau_1, \tau_2\}$ with $p(\tau_1) \leq p(\tau_2)$ is RM-schedulable if and only if

$$c(\tau_2) \leq \left\lfloor \frac{p(\tau_2)}{p(\tau_1)} \right\rfloor (p(\tau_1) - c(\tau_1)) + \max \left\{ 0, p(\tau_2) - \left\lfloor \frac{p(\tau_2)}{p(\tau_1)} \right\rfloor p(\tau_1) - c(\tau_1) \right\}.$$

This constant time test will be used in our matching based algorithm in Chapter 5. Note that for all constant $n$ the response times can be computed in polynomial time by Lenstra's algorithm for integer programming in fixed dimension [Len83].

For general $n$, Baruah & Fisher [FB05] obtained an FPTAS for approximating the processor speed, from which on the task system is RM-schedulable. To understand their result we again inspect the response time function for a task $\tau$ w.r.t. a task set $\mathcal{S}$

$$D_{\tau,\mathcal{S}}(r) = c(\tau) + \sum_{\tau' \prec \tau} \left\lceil \frac{r}{p(\tau')} \right\rceil \cdot c(\tau').$$

The idea is now to replace the roundup operation, applied to $r/p(\tau')$ by a $+1$ as soon as the fraction exceeds $1/\varepsilon$ for an error parameter $\varepsilon > 0$. Thus for

$$\delta(r, \tau') = \begin{cases} \lceil r/p(\tau') \rceil & \text{if } r/p(\tau') \leq 1/\varepsilon \\ r/p(\tau') + 1 & \text{otherwise} \end{cases}$$

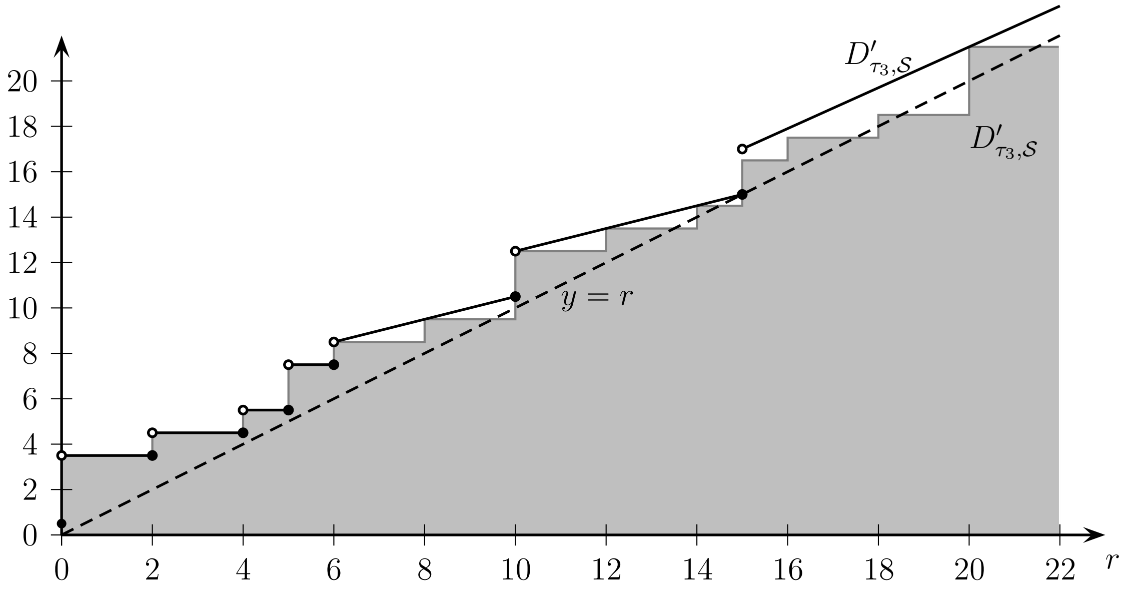


Figure 2.5: Piecewise linear approximation $D'_{\tau_3,\mathcal{S}}(r)$ (in black) w.r.t. the response time function $D_{\tau_3,\mathcal{S}}(r)$ (in gray) for $\tau_1 = (1,2), \tau_2 = (2,5), \tau_3 = (1/2, 12)$ and $\varepsilon = 1/3$. The bisecting line is drawn dashed. The approximate test fails to certify the feasibility of $\tau_3$.

we define a function

$$D'_{\tau,\mathcal{S}}(r) = c(\tau) + \sum_{\tau' \prec \tau} \delta(r, \tau') \cdot c(\tau'),$$

which is dominating the response time function, but never exceeds it by a factor of more than $1 + \varepsilon$, i.e. $D'_{\tau,\mathcal{S}}(r)/(1+\varepsilon) \leq D_{\tau,\mathcal{S}}(r) \leq D'_{\tau,\mathcal{S}}(r)$ for all $r \geq 0$. Furthermore the dominating function $D'_{\tau,\mathcal{S}}$ is piecewise linear with $\mathcal{O}(n/\varepsilon)$ many different line segments, thus the smallest intersection with the bisecting line can be easily computed in polynomial time. We obtain a test, which yields *no* if $\mathcal{S}$ is not feasible and *yes* if the set is feasible even after the running times are multiplied by $1 + \varepsilon$. However for task sets, that are barely feasible, the test might fail. Figure 2.5 shows an example.

A very useful quantity is the *utilization* $u(\tau) = \frac{c(\tau)}{p(\tau)}$, denoting the average fraction of processor cycles, which are used for $\tau$. We further define $u(\mathcal{S}) = \sum_{\tau \in \mathcal{S}} u(\tau)$ as the utilization of the whole task system. It is not difficult to see that if $u(\mathcal{S}) > 1$, the system cannot be schedulable on a single processor. On the other hand a system should be schedulable if the utilization is not too large. And in fact, Liu & Layland [LL73] have shown that if $u(\mathcal{S}) \leq n \cdot (\sqrt[n]{2} - 1)$ then $\mathcal{S}$ is RM-schedulable. This quantity is monotonically decreasing in $n$ and tends to $\ln(2) \approx 0.69$, see Figure 2.6.

However, this test is sufficient, but not necessary as the example from Figure 2.3 with a utilization of $u(\{\tau_1, \tau_2\}) = 90\%$ shows. The *hyperbolic test* of Bini et al. [BBB01] (see also [Liu00]) guarantees feasibility for $\mathcal{S}$ if

$$\prod_{\tau \in \mathcal{S}} (1 + u(\tau)) \leq 2$$

Taking the logarithm on both sides and using that $\ln(1 + x) < x$ for $x > 0$ we see that this test is strictly more accurate than that of [LL73], but still not necessary.

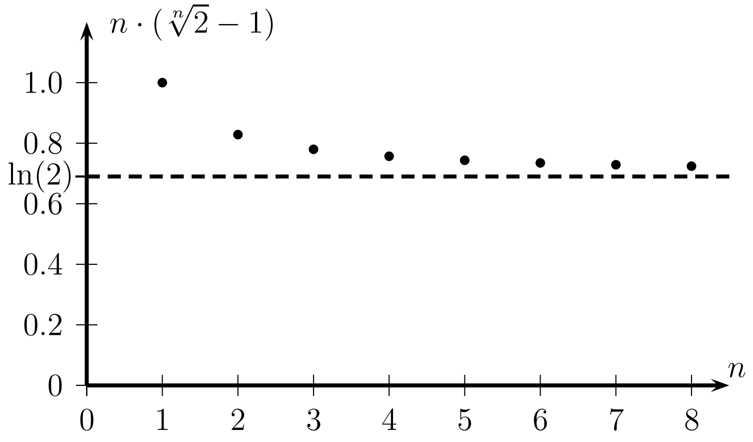


Figure 2.6: Visualisation of the Liu & Layland utilization bound

To obtain better criteria we need to incorporate also the relationship of the periods, since the result of Liu and Layland is tight in the sense, that for any $n \in \mathbb{N}$ and $\varepsilon > 0$, there is a task system consisting of $n$ tasks with total utilization $n \cdot (\sqrt[n]{2} - 1) + \varepsilon$, which is not RM-schedulable.

Suppose for example that all periods were pairwise divisible. We want to derive a condition, when a task $\tau$ is then feasible. Consider the response time function, divided by $p(\tau)$. We claim that $p(\tau)$ is a certificate for feasibility of $\tau$. Then

$$\frac{D_{\tau,\mathcal{S}}(p(\tau))}{p(\tau)} = \frac{c(\tau)}{p(\tau)} + \sum_{\tau' \prec \tau} \left\lceil \frac{p(\tau)}{p(\tau')} \right\rceil \cdot \frac{c(\tau')}{p(\tau)} = u(\tau) + \sum_{\tau' \prec \tau} u(\tau') \leq u(\mathcal{S}). \qquad (2.2)$$

using that $p(\tau') \mid p(\tau)$. We see that $\mathcal{S}$ must be feasible if the utilization is bounded by 1 – the best we can hope for. It seems natural to hope for similar, possibly weaker results in case that the periods are nearly multiples of each other.

Thus define

$$\alpha(\tau) = \log_2 p(\tau) - \lfloor \log_2 p(\tau) \rfloor$$

and

$$\beta(\mathcal{S}) = \max_{\tau \in \mathcal{S}} \alpha(\tau) - \min_{\tau \in \mathcal{S}} \alpha(\tau).$$

for a set of tasks $\mathcal{S}$. Clearly $0 \leq \alpha(\tau) < 1$. Furthermore the smaller $\beta(\mathcal{S})$ is, the closer are the periods to be multiples of each other. Note that the reverse does not necessarily hold, since for tasks $\tau_1, \tau_2$ with $p(\tau_1) = 2$ and $p(\tau_2) = 2 - \varepsilon$, one has $\beta(\{\tau_1, \tau_2\}) = \log_2(2 - \varepsilon) \approx 1$. On the other hand scaling the periods and running times decreases the $\beta$-value to almost 0, while not affecting the feasibility itself.

We now want to cite the following useful result from Burchard et al. [BLOS95] (see also [Leu04]).

**Theorem 2.1** ([BLOS95])**.** *Given an implicit-deadline task system $\mathcal{S}$, let $\beta := \beta(\mathcal{S}) = \max_{\tau \in \mathcal{S}} \alpha(\tau) - \min_{\tau \in \mathcal{S}} \alpha(\tau)$. Then*

*1. $\mathcal{S}$ is RM-schedulable if*

$$u(\mathcal{S}) \leq g_n(\beta) := \begin{cases} (n-1)(2^{\beta/(n-1)} - 1) + 2^{1-\beta} - 1 & \textit{if } \beta < 1 - \frac{1}{n} \\ n(2^{1/n} - 1) & \textit{else} \end{cases}$$

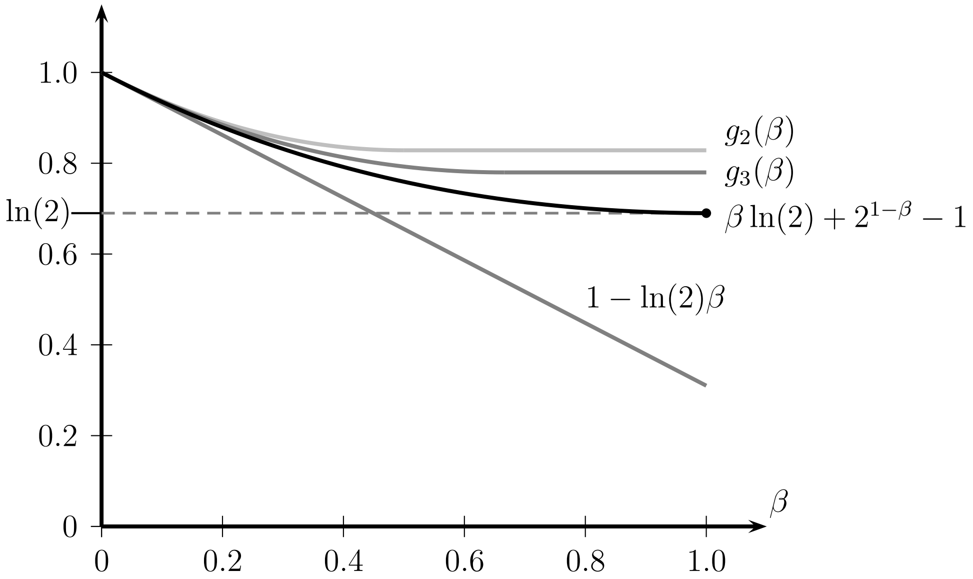


Figure 2.7: Utilization bounds for Burchard et al.'s feasibility criterion.

*2. $\mathcal{S}$ is RM-schedulable if*

$$u(\mathcal{S}) \leq \beta\ln(2) + 2^{1-\beta} - 1$$

*3. $\mathcal{S}$ is RM-schedulable if*

$$u(\mathcal{S}) \leq 1 - \beta\ln(2)$$

Thereby (2) follows by taking the limit for $n \to \infty$ of (1). Condition (3) is then a linear approximation to (2) for $\beta = 0$. These sufficient bounds on the utilization are visualized in Figure 2.7.

Note that even the best bound in Theorem 2.1 is not necessary for feasibility.

## 2.3 Multiprocessor scheduling

Again we consider RM-schedules for implicit-deadline tasks. The challenging goal here is: Given a set of tasks, distribute them on as few processors as possible.

Rate-monotonic Multiprocessor Scheduling (MulSched)
Given: A set $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$ of periodic tasks $\tau_i = (c(\tau_i), p(\tau_i))$ with implicit deadlines.
Find: Partition $\mathcal{S} = \mathcal{S}_1 \cup \ldots \cup \mathcal{S}_k$ s.t. each $\mathcal{S}_i$ is RM-schedulable and $k$ is minimized.

For the sake of completeness note that

**Definition 1.** Let $\rho \geq 1$. For a minimization problem $L$, $A$ is called a *$\rho$-approximation algorithm* if it has polynomial running time and

$$A_L(\mathcal{I}) \leq \rho \cdot OPT_L(\mathcal{I})$$

for all instances $\mathcal{I}$. Here, $OPT_L(\mathcal{I})$ gives the value of an optimum solution for instance $\mathcal{I}$ and $A_L(\mathcal{I})$ denotes the value of the solution, obtained by algorithm $A$.

We will omit $L$ and $\mathcal{I}$, if the problem and the instance, respectively, are clear from the context.

**Definition 2.** For a minimization problem $L$, $A$ is called an *asymptotic $\rho$-approximation algorithm*, if $A$ is a polynomial time algorithm, such that

$$\lim_{k\to\infty} \sup_{\mathcal{I}:OPT_L(\mathcal{I})\geq k} \left\{ \frac{A_L(\mathcal{I})}{OPT_L(\mathcal{I})} \right\} \leq \rho.$$

For example, an algorithm obtaining solutions of cost at most $2 \cdot OPT + \sqrt{OPT} + 4$ is an asymptotic 2-approximation algorithm.

Till now most MulSched algorithms work similar to simple BinPacking heuristics (see also Section 2.4): They sort tasks w.r.t. a specific criterion and then distribute them either in a First Fit or in a Next Fit manner, using a sufficient feasibility test. Here, *Next Fit manner*, means an active processor is maintained and incoming tasks are assigned to it, until the feasibility criterion fails. Then a new processor is opened. Old processor are never considered again. Such algorithms have the advantage, that they are very simple – at the expense of the quality, i.e. the number of processor. For First Fit, tasks are assigned to the first processor, on which they fit, according to the feasibility criterion. Here, the *first* processor is that one with the smallest index. The following table gives an overview over popular algorithms from the literature, see [Dha04].

| **Name** | **sorting** | **distribution** | **ratio** | **run time** |
|---|---|---|---:|---|
| `RMNF` | inc. periods | Next Fit | 2.67 | $\mathcal{O}(n\log n)$ |
| `RMFF` | inc. periods | First Fit | 2.00 | $\mathcal{O}(n\log n)$ |
| `FFDU` | dec. utilization | First Fit | 2.00 | $\mathcal{O}(n\log n)$ |
| `RMST` | inc. $\alpha(\tau)$ | Next Fit | $\frac{1}{1-\max_{\tau\in\mathcal{S}} u(\tau)}$ | $\mathcal{O}(n\log n)$ |
| `RMGT` | - | First Fit+`RMST` | 1.75 | $\mathcal{O}(n^2)$ |
| `FFMP` | inc. $\alpha(\tau)$ | First Fit | 2.00 | $\mathcal{O}(n\log n)$ |
| `RMMatching` | - | matching+`FFMP` | 1.50 | $\mathcal{O}(n^3)$ |

For the sake of comparison, we list two algorithms `FFMP` and `RMMatching` in the last rows, which we are going to introduce in Chapters 4 and 5. The ratio column gives the bound on the asymptotic approximation ratio. The *Rate-monotonic general task* algorithm distributes tasks with utilization at most $1/3$ using `RMST` and the rest separately with First Fit. A more detailed description can be found in [Leu04]. One observes that `RMST` is nearly optimal for tasks with small utilization. The used sorting principle will be useful for us in Chapter 4.

## 2.4 Bin Packing

Suppose we are given an instance $\mathcal{S}$ of implicit-deadline tasks and all periods were pairwise divisible, then $\mathcal{S}$ would be RM-schedulable on a single processor if and only if $u(\mathcal{S}) \leq 1$. This condition reminds us of the famous BinPacking problem.

BinPacking
Given: A set of items $I = \{1,\ldots,n\}$ with item sizes $a_i \in [0,1]$
Find: Partition of $I$ into *bins* $I_1,\ldots,I_k$ such that $\sum_{i\in I_j} a_i \leq 1$ for all $j$ and $k$ is minimized.

Let us denote size$(I) = \sum_{i \in I} a_i$, then we see that BinPacking is a special case of MulSched, where each item $i$ corresponds to a task, whose utilization equals the size $a_i$.

Three classical algorithms for BinPacking are `FirstFit`, `NextFit` and `BestFit`, see Johnson [Joh73]. In each algorithm one starts with empty bins and distributes the items consecutively.

- `FirstFit`: Assign the current item to the first bin (i.e. that one with a minimal index), having enough space.
- `NextFit`: Maintain an active bin. Assign the current item to the active bin if possible, otherwise assign it to an empty bin and let this bin be the active one.
- `BestFit`: Assign the current item to the fullest bin with enough space.

`NextFit` gives a 2-approximation since two consecutive bins must have a total size of at least 1. `FirstFit` produces solutions with not more than $1.7 \cdot OPT + 1$ bins [JDU$^+$74]. If items are initially sorted w.r.t. their sizes in non-decreasing order, `FirstFit` even needs only $\frac{11}{9} OPT + \frac{6}{9}$ bins [Dós07]. This algorithm is called `FirstFitDecreasing` [JDU$^+$74]. `BestFitDecreasing` gives the same asymptotic ratio of $\frac{11}{9}$. More on BinPacking results can be found in the excellent survey of Coffman, Garey & Johnson [CGJ84].

The question whether 2 bins suffice or 3 are needed, is exactly the so called Partition problem, which is weakly **NP**-hard [Joh92, Weg05], but solvable via dynamic programming in pseudo-polynomial time.

Partition
Given: A multi-set of numbers $a_1, \ldots, a_n$.
Decide: Is there a partition $I_1 \dot\cup I_2 = \{1, \ldots, n\}$ such that

$$\sum_{i \in I_1} a_i = \sum_{i \in I_2} a_i$$

We conclude that unless $\mathbf{NP} \neq \mathbf{P}$, for any $\varepsilon > 0$ there cannot be a $(3/2 - \varepsilon)$-approximation algorithm for BinPacking, since such an algorithm would solve the Partition problem in polynomial time. Thus a PTAS cannot exist for BinPacking. On the other hand, a polynomial time algorithm distinguishing, say, $OPT_{\text{BinPacking}} \leq 20$ and $OPT_{\text{BinPacking}} \geq 30$ is easy to obtain (see e.g. [Vaz01]). Consequently it makes sense to consider the achievable approximation ratio for large values of $OPT_{\text{BinPacking}}$.

**Definition.** [Vaz01] Algorithm $A_\varepsilon$ is termed an *asymptotic polynomial time approximation scheme (PTAS)* for minimization problem $L$, if for any fixed $\varepsilon > 0$, it runs in time polynomial in $|\mathcal{I}|$ and

$$A_\varepsilon(\mathcal{I}) \leq (1 + \varepsilon) \cdot OPT(\mathcal{I})$$

for all instances $\mathcal{I}$ with $OPT(\mathcal{I})$ large enough.

Such an asymptotic PTAS for BinPacking was found by Fernandez de la Vega and Lueker [FdlVL81]. Considering the running time of $n^{\mathcal{O}(1/\varepsilon)^{\mathcal{O}(1/\varepsilon)}}$, it is not hard to believe that this algorithm is mainly intended to be a theoretical result. An algorithm that also behaves well in practice should also have polynomial running time in $1/\varepsilon$. This motivates the following definition.

**Definition.** Let $\varepsilon > 0$. Algorithm $A_\varepsilon$ is called an *asymptotic FPTAS* if

$$A_\varepsilon(\mathcal{I}) \leq (1+\varepsilon) \cdot OPT(\mathcal{I}) + p(1/\varepsilon)$$

for all instances $\mathcal{I}$, whereby $p$ is a polynomial and the running time is polynomial in $n$ and in $1/\varepsilon$.

Karmarkar & Karp [KK82] obtained such an asymptotic FPTAS for BinPacking, using the Gilmore-Gomory LP relaxation. More on this topic can be found in Chapter 6.

Let us close this introduction on BinPacking with a comparison of all popular algorithms. The approximation ratios in the following table are meant asymptotically, i.e. for $OPT \to \infty$.

| **Name** | **year** | **ratio** | **run time** |
|---|---:|---:|---:|
| `NextFit` | 1973 [Joh73] | 2 | $\mathcal{O}(n)$ |
| `FirstFit` | 1972 [GGU72, JDU+74] | 17/10 | $\mathcal{O}(n \log n)$ |
| `BestFit` | 1973 [Joh73, JDU+74] | 17/10 | $\mathcal{O}(n \log n)$ |
| `FirstFitDecreasing` | 1973 [Joh73, Dós07] | 11/9 | $\mathcal{O}(n \log n)$ |
| `BestFitDecreasing` | 1973 [Joh73] | 11/9 | $\mathcal{O}(n \log n)$ |
| F. de la Vega/Lueker | 1981 [FdlVL81] | $1+\varepsilon$ | $n^{\mathcal{O}(1/\varepsilon)^{\mathcal{O}(1/\varepsilon)}}$ |
| Karmarkar/Karp | 1982 [KK82] | $1+\varepsilon$ | poly$(n, 1/\varepsilon)$ |

## 2.5 Related scheduling problems

To get a broader overview, we discuss other combinations of deadline-types and scheduling policies.

### 2.5.1 Dynamic priority scheduling & implicit deadlines

Suppose that we are not forced any more to assign fixed priorities, thus priorities may depend on time and EDF is an optimal scheduler [Der74]. Already Liu & Layland [LL73] have shown that a set $\mathcal{S}$ of implicit-deadline tasks is EDF-schedulable if and only if $u(\mathcal{S}) \leq 1$, thus the schedulability test is polynomial and multiprocessor scheduling coincides with BinPacking.

### 2.5.2 Fixed priorities & constrained deadlines

Next, consider the case that tasks have constrained deadlines, i.e. $d(\tau) \leq p(\tau)$. Recall that here Deadline-monotonic priorities $\frac{1}{d(\tau)}$ are optimal [LW82]. As for implicit-deadline tasks, the critical instance is the first job, released at time 0 (which is not the case anymore for arbitrary deadlines, see [Leh90]). Due to this reason, the response time can again be obtained as minimal solution of fix-point equation (2.1). Note that even if the utilization is arbitrarily small, the system might not be schedulable on a single machine. For example the task system $\tau_1 = \tau_2 = (1, 1, M)$ (using notation $\tau = (c(\tau), d(\tau), p(\tau))$) is not DM-schedulable for any $M$ although the utilization is just $2/M$.

### 2.5.3 Dynamic priorities & arbitrary deadlines

Finally suppose that we have to schedule explicit deadline tasks with a dynamic priority schedule. The result of Dertouzos [Der74] applies also here, thus EDF-scheduling is optimal. But again the condition $u(\mathcal{S}) \leq 1$ is not sufficient anymore. To see this, consider again a task system consisting of $\tau_1 = \tau_2 = (1, 1, M)$. Then in the interval from 0 to 1, jobs of both tasks must be completed. Since the sum of the running times is 2, this is impossible, regardless to the used schedule. The *demand bound function*

$$\texttt{DBF}(\mathcal{S}, t) = \sum_{\tau \in \mathcal{S}} \max \left\{ \left( \left\lfloor \frac{t - d(\tau)}{p(\tau)} \right\rfloor + 1 \right), 0 \right\} \cdot c(\tau)$$

gives the running time of all jobs, which have their release time and deadline in $[0, t]$. Thus for feasibility it is necessary, that $\texttt{DBF}(\mathcal{S}, t) \leq t$ for all $t \geq 0$. Baruah et al. [BMR90] showed that this condition is in fact sufficient. Given that $\mathcal{S}$ is not EDF-schedulable, the smallest $t > 0$, certifying the infeasibility must have

$$t < \frac{u(\mathcal{S})}{1 - u(\mathcal{S})} \max_{\tau \in \mathcal{S}} \{p(\tau) - c(\tau)\},$$

see e.g. [BG04, Leu04]. This admits a pseudo-polynomial algorithm for the feasibility test, if the utilization of $\mathcal{S}$ is bounded away from 1 by a constant.

A quantity, which is useful in this setting is the *load*, defined as

$$\texttt{LOAD}(\mathcal{S}) = \max_{t > 0} \left\{ \frac{\texttt{DBF}(\mathcal{S}, t)}{t} \right\}$$

This maximum is clearly attained. Then $\mathcal{S}$ is EDF-schedulable if and only if $\texttt{LOAD}(\mathcal{S}) \leq 1$. But we will prove in Chapter 8, that testing this condition is **coNP**-hard even if the tasks have constrained deadlines and the demand bound function can be simplified to

$$\texttt{DBF}(\mathcal{S}, t) = \sum_{\tau \in \mathcal{S}} \left( \left\lfloor \frac{t - d(\tau)}{p(\tau)} \right\rfloor + 1 \right) \cdot c(\tau).$$

In Chapter 10 we give an exact quantification of the power of EDF-scheduling for constrained-deadline tasks compared to the DM-schedule.

## 2.6 Notation

For the sake of completeness we now define some notation. We denote $\mathbb{N} = \{1, 2, \ldots\}$, i.e. 0 is not included. For $\mathbb{Q}_+$ is the set of strictly positive numbers, while $\mathbb{Q}_{\geq 0}$ is the set of non-negative rationals. $\mathbf{0} = (0, \ldots, 0)$ and $\mathbf{1} = (1, \ldots, 1)$ denote suitably sized all-0 and all-1 vectors, respectively. $\log(x) := \log_2(x)$ always denotes the logarithm w.r.t. base 2, $\ln(x)$ gives the logarithm of $x$ w.r.t. base $e$. We define the norms

$$\|v\|_p := \left( \sum_{i=1}^{n} |v_i|^p \right)^{1/p} \qquad \text{and} \qquad \|v\|_\infty := \max_{i=1,\ldots,n} |v_i|$$

for a vector $v \in \mathbb{R}^n$ and $1 \leq p < \infty$.

The names of problems will be written in Capitals, while algorithms are written in `TypeWriter`.

# Chapter 3

# An asymptotic PTAS under Resource Augmentation

In this chapter, we deal with approximating MulSched, i.e. we want to partition a set $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$ of implicit-deadline tasks into sets $\mathcal{S}_1, \ldots, \mathcal{S}_k$ such that each $\mathcal{S}_i$ is RM-schedulable and $k$ is minimized. But we face the problem, that even if an assignment of the tasks to the processors is given, we would not know, whether we could certify the feasibility in polynomial time. On the other hand, the test of Baruah & Fisher [FB05] allows to certify the feasibility, after the processor is speeded up by a factor of $1 + \varepsilon$. Here we aim at a generalization of this result from the single processor to the multiprocessor case. We will prove that given a fixed $\varepsilon > 0$ and a set $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$ of implicit-deadline tasks, we can find an assignment to processors $\mathcal{S}_1, \ldots, \mathcal{S}_k$ with $k \leq (1 + \varepsilon)OPT_{\textsc{MulSched}}(\mathcal{S}) + \mathcal{O}(1)$, such that the RM-schedule of each $\mathcal{S}_j$ is feasible if the running times are divided by $1+\varepsilon$. In other words, we obtain an asymptotic PTAS under resource augmentation. This chapter is borrowed from Eisenbrand & Rothvoß [ER08a].

Since MulSched is a generalisation of BinPacking, we should naturally ask, what we would do to obtain a PTAS in that special case. The asymptotic BinPacking PTAS of Fernandez de la Vega & Lueker [FdlVL81, Vaz01] works as follows: First discard the small items with a size of at most $\varepsilon$. Round up the sizes such that just a constant number of different item sizes remain and $OPT_{\textsc{BinPacking}}$ increases by a factor of not more than $1 + \varepsilon$. Then each bin must be packed with a constant number of items, chosen from a candidate set of constant cardinality. Consequently the number of possibilities, how a single bin can be packed is bounded by a (huge) constant and the number of solutions that have to be considered is just a polynomial, though the exponent is astronomically large. Afterwards the initially discarded small items are distributed in a First Fit manner over the already packed bins. If no new bin needs to be opened for the small items, the approximation guarantee directly follows. Otherwise all but one bin must be filled up to at least $1 - \varepsilon$, thus the solution is near optimal as well.

Unfortunately for MulSched such a treatment of tasks with small utilization cannot work, since in the case that the utilization of a processor exceeds 69%, it may be that even a task with tiny utilization $\varepsilon > 0$ could not be added without making the set infeasible.

Back to BinPacking, if we may slightly overpack a bin, a different approach might work as follows: We initially glue together small items and round *down* item sizes s.t. a constant number of different item types remain. The emerging instance is *dominating* in the sense that the modified items reserve enough space for the original items, at least if each bin may be filled up to a value of $1+\varepsilon$, instead of 1. By generalizing this approach for MulSched we obtain tasks whose utilization is a multiple of some constant (depending on $\varepsilon$).

The remaining challenge is that in general the periods range over a large interval, thus we may not assume to have just a constant number of different task types. We will resolve this problem by introducing a relaxed notion for feasibility, which allows to compute a solution via dynamic programming.

Throughout this chapter, a constant independent from $\varepsilon$ is denoted by $\mathcal{O}(1)$ and $\mathcal{O}_\varepsilon(1)$ otherwise.

## 3.1 Domination

Now let us extend the concept of domination to implicit-deadline task systems $\mathcal{S}$. We say that task system $\mathcal{S}'$ *dominates* $\mathcal{S}$, if there is a map $\pi : \mathcal{S} \to \mathcal{S}'$ such that for any feasible MulSched solution $\mathcal{S}' = \mathcal{S}'_1 \dot\cup \ldots \dot\cup \mathcal{S}'_k$ also all sets $\pi^{-1}(\mathcal{S}'_i)$ are RM-schedulable. Here $\pi$ is not necessarily injective, that means we allow that several tasks may be bundled in the dominating instance $\mathcal{S}'$.

It immediately follows that

**Corollary 3.1.** *If $\mathcal{S}'$ dominates $\mathcal{S}$, then*

$$OPT_{\text{MulSched}}(\mathcal{S}) \leq OPT_{\text{MulSched}}(\mathcal{S}').$$

In fact, a solution for $\mathcal{S}'$ of value $k$ directly gives a solution for $\mathcal{S}$ with the same value. In all constructions of dominating instances the function $\pi$ will be described implicitly, but can be computed in polynomial time. But for the sake of readability we will never state $\pi$ explicitly.

The following operations yield dominating instances:

- Replace a task $\tau$ by another task $\tau'$ with $c(\tau') \geq c(\tau)$ and $p(\tau') \leq p(\tau)$.
- Replace tasks $\tau_1, \ldots, \tau_m \in \mathcal{S}$ of the same period (i.e. $p(\tau_1) = \ldots = p(\tau_m)$) by a single task $\tau'$ with accumulated running time $c(\tau') = c(\tau_1) + \ldots + c(\tau_m)$ and period $p(\tau') = p(\tau_1) = \ldots = p(\tau_m)$.

For the sake of approximation algorithms a more general notion of domination will turn out to be helpful.

**Definition.** Given a task system $\mathcal{S}$ and an $\varepsilon > 0$, the system $\mathcal{S}'$ is *weakly $\varepsilon$-dominating* $\mathcal{S}$, if there is a polynomial time computable function $\pi : \mathcal{S} \to \mathcal{S}'$, such that for any feasible MulSched solution $\mathcal{S}' = \mathcal{S}'_1 \dot\cup \ldots \dot\cup \mathcal{S}'_k$, there is a polynomial time computable set $\mathcal{S}^d$ of discarded tasks with $u(\mathcal{S}^d) \leq \varepsilon \cdot u(\mathcal{S})$, such that all $\pi^{-1}(\mathcal{S}'_1) \backslash \mathcal{S}^d, \ldots, \pi^{-1}(\mathcal{S}'_k) \backslash \mathcal{S}^d$ are RM-schedulable.

We should give the reader some intuition about the meaning of this definition: Suppose we run an algorithm on $\mathcal{S}'$, obtain a solution $\mathcal{S}'_1, \ldots, \mathcal{S}'_k$ and compute the corresponding solution $\mathcal{S}_1, \ldots, \mathcal{S}_k$ in the original instance $\mathcal{S}$. For a nearly optimal solution it is then sufficient if $\mathcal{S}_1, \ldots, \mathcal{S}_k$ are not RM-schedulable, but can be made feasible after discarding an $\varepsilon$-fraction of the tasks.

For discarding tasks we should first observe a simple claim, which suffices for our purposes.

**Lemma 3.2.** *Given a set $\mathcal{S}$ of implicit-deadline tasks, one can efficiently find a solution for* MulSched*, consisting of at most $3u(\mathcal{S}) + 1$ many processors.*

*Proof.* We distribute the tasks in a First Fit manner, considering a set $\mathcal{S}' \subseteq \mathcal{S}$ as feasible if $u(\mathcal{S}') \leq \ln(2)$ (cf. [LL73]). Let $m$ be the number of needed processors, then all but (possibly) one processor must have a utilization of at least $\ln(2)/2 > 1/3$, otherwise we could merge processors. We derive the inequality $u(\mathcal{S}) \geq (m-1)/3$ and hence $m \leq 3u(\mathcal{S}) + 1$. □

Here we did not aim at optimizing any constant. Next, we demonstrate the usefulness of the notion of weak $\varepsilon$-domination.

**Corollary 3.3.** *If $\mathcal{S}'$ weakly $\varepsilon$-dominates $\mathcal{S}$, then a solution of cost $k$ for $\mathcal{S}'$ can be efficiently turned into a feasible solution for $\mathcal{S}$ of cost at most $k + 3\varepsilon \cdot u(\mathcal{S}) + 1$.*

*Proof.* Let $\mathcal{S}'_1, \ldots, \mathcal{S}'_k$ be a solution for $\mathcal{S}'$. Let $\mathcal{S}^d$ be the discarded tasks such that each $\mathcal{S}_i = \pi^{-1}(\mathcal{S}'_i) \backslash \mathcal{S}^d$ is feasible. Lemma 3.2 yields, that $\mathcal{S}^d$ can be scheduled on

$$3u(\mathcal{S}^d) + 1 \leq 3\varepsilon \cdot u(\mathcal{S}) + 1$$

many processors. The claim follows. □

## 3.2 Relaxing the feasibility

When designing an approximation algorithm for MulSched we face the feasibility test as a difficult subproblem. Next, we develop a relaxed feasibility test, which is similar to that one of [FB05], but tailored for our dynamic programming approach (see Section 3.4). Recall that an implicit-deadline task system $\mathcal{S}$ is RM-schedulable if and only if for any task $\tau \in \mathcal{S}$ one has an $r \in [0, p(\tau)]$ such that

$$c(\tau) + \sum_{\tau' \prec \tau} \left\lceil \frac{r}{p(\tau')} \right\rceil \cdot c(\tau') \leq r. \tag{3.1}$$

We would like to relax this condition in a suitable way. Let us consider the left hand side of inequality (3.1). Given any solution $r$ and $z \in \mathbb{N}$, also $z \cdot r$ solves the inequality since

$$\left\lceil \frac{zr}{p(\tau')} \right\rceil \leq z \cdot \left\lceil \frac{r}{p(\tau')} \right\rceil$$

Thus we may assume that a solution to (3.1) fulfills $r \in [p(\tau)/2, p(\tau)]$, even if this $r$ then is not the response time (but an integer multiple of it). Now suppose that the left hand side of (3.1) contains a summand $\lceil \frac{r}{p(\tau')} \rceil$ with $p(\tau') \ll p(\tau)$. Then consequently $p(\tau') \ll r$. But if we do not round up a large value $\frac{r}{p(\tau')} \gg 1$, then we change the response time function just by a small factor. This motivates the upcoming definition of *local feasibility.*

**Definition.** Given an implicit-deadline task system $\mathcal{S}$, a task $\tau \in \mathcal{S}$ is called *locally feasible* w.r.t. $0 < \varepsilon < 1$, if there is an $r \in [0, p(\tau)]$ with

$$D^{\varepsilon}_{\tau,\mathcal{S}}(r) = c(\tau) + \sum_{\tau' : p(\tau') \leq \varepsilon p(\tau)} \frac{r}{p(\tau')} \cdot c(\tau') + \sum_{\tau' \prec \tau : \varepsilon p(\tau) < p(\tau') \leq p(\tau)} \left\lceil \frac{r}{p(\tau')} \right\rceil \cdot c(\tau') \leq r$$

The task system $\mathcal{S}$ is called locally feasible, if all its tasks are locally feasible.

Note that $D^{\varepsilon}_{\tau,\mathcal{S}}(r)$ can be rewritten as

$$c(\tau) + r \cdot u(\{\tau' \in \mathcal{S} \mid p(\tau') \leq \varepsilon \cdot p(\tau)\}) + \sum_{\tau' \prec \tau : p(\tau') > \varepsilon p(\tau)} \left\lceil \frac{r}{p(\tau')} \right\rceil \cdot c(\tau')$$

In other words, the contribution of the rounding operation is only taken into account for tasks which are *close* or *local* to the task $\tau$ in consideration. The other tasks contribute only with their utilization. This relaxed feasibility notion causes a drastic reduction of complexity, as we will see in Section 3.4.

We now show that, if an assignment is locally feasible, then it is "really" feasible (i.e. RM-schedulable) on a slightly faster processor.

**Lemma 3.4.** *If a set of tasks $\mathcal{S}$ is locally feasible w.r.t. $\varepsilon > 0$, then it is feasible on a processor of speed $1 + 2\varepsilon$.*

*Proof.* Let $r$ be the certificate for local feasibility of $\tau \in \mathcal{S}$, i.e., one has $p(\tau)/2 \leq r \leq p(\tau)$ and $D^{\varepsilon}_{\tau,\mathcal{S}}(r) \leq r$. It is enough to show that $D_{\tau,\mathcal{S}}(r) \leq (1 + 2\varepsilon) D^{\varepsilon}_{\tau,\mathcal{S}}(r)$ holds. We derive that

$$\begin{aligned} D_{\tau,\mathcal{S}}(r) - D^{\varepsilon}_{\tau,\mathcal{S}}(r) &\leq \sum_{\tau' \prec \tau : p(\tau') \leq \varepsilon p(\tau)} c(\tau') \\ &\leq 2\varepsilon r \cdot \sum_{\tau' \prec \tau : p(\tau') \leq \varepsilon p(\tau)} \frac{c(\tau')}{p(\tau')} \\ &\leq 2\varepsilon \cdot D^{\varepsilon}_{\tau,\mathcal{S}}(r) \end{aligned}$$

where we use that $\frac{r}{p(\tau')} \geq \frac{r}{\varepsilon p(\tau)} \geq \frac{1}{2\varepsilon}$ and consequently $1 \leq 2\varepsilon \frac{r}{p(\tau')}$. □

More general for $\delta \geq 0$, we call a task $\tau \in \mathcal{S}$ *locally* $(1 + \delta)$*-feasible*, if there is an $r \leq p(\tau)$ such that $D^{\varepsilon}_{\tau,\mathcal{S}}(r) \leq (1 + \delta)r$ or equivalently $\tau$ is locally feasible on a processor of speed $1 + \delta$. We term a system $\mathcal{S}$ $(1+\delta)$*-feasible*, if it is RM-schedulable on a processor of speed $1 + \delta$. Clearly

**Corollary 3.5.** *For $\delta \geq 0$, if a set of tasks $\mathcal{S}$ is locally $(1 + \delta)$-feasible on one processor, then it is feasible on a processor of speed $(1 + 2\varepsilon) \cdot (1 + \delta)$.*

*Proof.* Suppose $p(\tau)/2 \leq r \leq p(\tau)$ and $D^{\varepsilon}_{\tau,\mathcal{S}}(r) \leq (1 + \delta)r$, then reusing the proof of Lemma 3.4 yields

$$D_{\tau,\mathcal{S}}(r) \leq (1 + 2\varepsilon) D^{\varepsilon}_{\tau,\mathcal{S}}(r) \leq (1 + 2\varepsilon) \cdot (1 + \delta) \cdot r.$$

□

## 3.3 The merging theorem

In this section we show that small tasks of similar periods can be merged together to obtain large tasks. We will obtain a weakly $\varepsilon$-dominating instance $\mathcal{S}'$, containing only tasks whose utilization is lower bounded by a constant and $OPT_{\text{MulSched}}(\mathcal{S}') \leq (1 + \mathcal{O}(\varepsilon))OPT_{\text{MulSched}}(\mathcal{S}) + \mathcal{O}(1)$. Here the relaxed feasibility notion will already be useful.

In the following we call a task *small* if its utilisation is at most $\gamma = \varepsilon^6$ and *large* otherwise. Let $\mathcal{S} = \mathcal{S}_\ell \cup \mathcal{S}_s$ be the partition of the task system $\mathcal{S}$ into small tasks $\mathcal{S}_s$ and large tasks $\mathcal{S}_\ell$.

The whole procedure to eliminate small tasks can be divided into 3 steps

I) *Clustering:* In a first step, we discard tasks and re-set periods such that the utilization of each period is at least $\varepsilon^6$. Here, the utilization of a period $p$ is the sum of the utilization of the tasks having period $p$. The total utilization of the discarded tasks is bounded by $\mathcal{O}(\varepsilon) \cdot u(\mathcal{S})$. Furthermore if $\mathcal{S}_i \subseteq \mathcal{S}$ is RM-schedulable and $\mathcal{S}'_i \subseteq \mathcal{S}'$ is the set of the modified tasks, then $\mathcal{S}'_i$ remains $(1 + \mathcal{O}(\varepsilon))$-feasible.

II) *Merging:* In a second step, we partition small tasks of the same period into groups, each of which will be identified into one single task having utilization of roughly $\varepsilon^6$. We show that this increases the value of $OPT$ only by a factor of $1 + \mathcal{O}(\varepsilon)$ if again a slight speedup of the processor speed is admitted.

III) *Post-processing*: Finally we prove, that it suffices to remove a tiny subset of the tasks from a $(1 + \mathcal{O}(\varepsilon))$-feasible system to establish again feasibility.

Now the steps are described in detail. Thereby suppose that $1/\varepsilon$ is an integer and $u(\mathcal{S}) \geq 1/2 \geq \varepsilon$. This assumptions are w.l.o.g. since in case of $u(\mathcal{S}) < 1/2$ one processor trivially suffices and the following machinery would not be needed.

In the following we will several times *discard* tasks. Let $\mathcal{S}^d$ be a set of such tasks, then we can partition them into RM-schedulable sets $\mathcal{S}_1^d, \ldots, \mathcal{S}_m^d$ with $m \leq 3 \cdot u(\mathcal{S}^d) + 1$ many processors (see Lemma 3.2). Since we are constructing a dominating instance, formally we replace each set of tasks $\mathcal{S}_i^d$ by a task with utilization 1 and arbitrary period. This ensures that in any feasible schedule these new tasks would use their own processor.

### Clustering

Let $p$ be a period of a task in $\mathcal{S}$. The *utilization* of this period is the sum of the utilizations of tasks, having period $p$

$$u(p) := \sum_{\tau \in \mathcal{S}: p(\tau) = p} u(\tau).$$

*Resetting* the period of a task $\tau$ to period $p$ means to substitute it by a task $\tau'$ which has period $p$ but the same utilization as $\tau$, i.e. $u(\tau') = u(\tau)$. Later we want to merge small tasks with the same period. In case that the instance is sparse, there are possibly no tasks with similar periods. Thus we need to apply the following clustering procedure to $\mathcal{S}$.

1. Round down the periods of the small tasks to the next power of $1 + \varepsilon$.

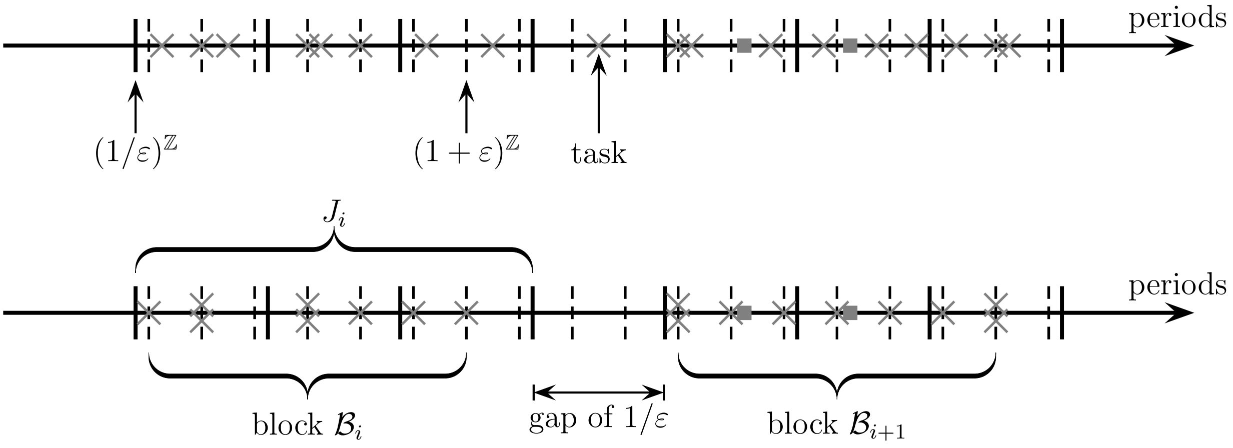

periods
$(1/\varepsilon)^{\mathbb{Z}}$
$(1+\varepsilon)^{\mathbb{Z}}$
task
$J_i$
periods
block $\mathcal{B}_i$
gap of $1/\varepsilon$
block $\mathcal{B}_{i+1}$

Figure 3.1: Consider the axis of real numbers and draw a × at $p(\tau)$ for each small task $\tau$ and a square for each large task. The upper picture visualizes the situation before step (1), the lower picture shows the partition into blocks (after step (3)).

2. Choose $j \in \{0, \ldots, (1/\varepsilon) - 1\}$ such that the utilization $u_j$ of tasks, having their period in an interval $[(1/\varepsilon)^i, (1/\varepsilon)^{i+1}[$ with $i \equiv_{1/\varepsilon} j$, is minimized. By the pigeonhole principle $u_j \leq \varepsilon \cdot u(\mathcal{S})$. We discard the tasks, contributing to $u_j$.

3. Partition the task set $\mathcal{S}$ into *blocks* $\mathcal{B}_1, \ldots, \mathcal{B}_m$ such that

   i) For $\tau \in \mathcal{B}_i, \tau' \in \mathcal{B}_j$ with $i < j$ one has $p(\tau) \leq \varepsilon \cdot p(\tau')$.

   ii) Due to the rounding, the number of different periods of small tasks in each block $\mathcal{B}_i$ is bounded by

   $$1 + \log_{1+\varepsilon}(1/\varepsilon)^{1/\varepsilon - 1} \leq 1/\varepsilon^3,$$

   which is a constant. See Figure 3.1 for a visualization. Each block may still contain large tasks with a numerous amount of different periods.

4. Let $\mathcal{B}_i$ be the first block having utilization at most $\varepsilon^2$, i.e. $u(\mathcal{B}_i) \leq \varepsilon^2$ for $i$ minimal. Then choose $j \geq i$ as the minimal index such that $u(\mathcal{B}_i \cup \ldots \cup \mathcal{B}_j) \geq \varepsilon^2$. If this utilization is larger than $\varepsilon$, then we discard $\mathcal{B}_i, \ldots, \mathcal{B}_{j-1}$ from $\mathcal{S}$. Otherwise, we reset the period of each task to an arbitrary value sandwiched between the smallest and the largest period of a task in $\mathcal{B}_i \cup \ldots \cup \mathcal{B}_j$. Thereby the utilization of this period is at least $\varepsilon^2$. We repeat this procedure, until such a block $\mathcal{B}_i$, having utilization at most $\varepsilon^2$, cannot be found anymore.

5. If there is a period $p$ such that the utilization of small tasks with period $p$ is less then $\varepsilon^6$, then we discard all small tasks with this period.

What is the sense of this clustering procedure: Now for each period $p$, the utilization of small tasks with that period is either 0 or lowerbounded by some constant.

Consider again a block $\mathcal{B}_i$. Due to the way, blocks are created we can assign an interval $J_i = [\frac{1}{\varepsilon^{i_1}}, \frac{1}{\varepsilon^{i_2}}]$ with $i_1, i_2 \in \mathbb{Z}$ to $\mathcal{B}_i$, such that for any $\tau \in \mathcal{B}_i$ one has $p(\tau) \in J_i$, the intervals are pairwise disjoint and all powers of $1/\varepsilon$ are covered by intervals[1]. The interval bounds of $J_i$ are then called the *period bounds* of $\mathcal{B}_i$. The period bounds of $\mathcal{B}_i, \ldots, \mathcal{B}_j$ $(i \leq j)$ then yield the interval from the lower period bound of $\mathcal{B}_i$ to the upper bound of $\mathcal{B}_j$.

Let $\mathcal{S}_i \subseteq \mathcal{S}$. We term a task $\tau \in \mathcal{B}_j \cap \mathcal{S}_i$ *locally feasible w.r.t. the block partition* if there is an $r \leq p(\tau)$ with

$$D_{\tau,\mathcal{S}_i}^{\mathcal{B}_1,\ldots,\mathcal{B}_m}(r) := c(\tau) + r \cdot u((\mathcal{B}_1 \cup \ldots \cup \mathcal{B}_{j-1}) \cap \mathcal{S}_i) + \sum_{\tau' \in \mathcal{B}_j \cap \mathcal{S}_i : \tau' \prec \tau} \left\lceil \frac{r}{p(\tau')} \right\rceil \cdot c(\tau') \leq r$$

Since $p(\tau) \leq \varepsilon p(\tau')$ for all $\tau \in \mathcal{B}_i, \tau' \in \mathcal{B}_j$ with $i < j$, the stated lemmas for local feasibility still hold for local feasibility w.r.t. the block partition. More precisely one has

$$D_{\tau,\mathcal{S}_i}^{\varepsilon}(r) \leq D_{\tau,\mathcal{S}_i}^{\mathcal{B}_1,\ldots,\mathcal{B}_m}(r) \leq D_{\tau,\mathcal{S}_i}(r) \leq (1+2\varepsilon) D_{\tau,\mathcal{S}_i}^{\varepsilon}(r)$$

for all $\tau \in \mathcal{S}_i$ and $p(\tau)/2 \leq r \leq p(\tau)$.

We next show that any $(1+\mathcal{O}(\varepsilon))$-feasible multiprocessor schedule is still $(1+\mathcal{O}(\varepsilon))$-feasible after the resetting procedure.

**Lemma 3.6.** *For $\delta \geq 0$ and $0 < \varepsilon \leq 1/2$, let $\mathcal{S}$ be a set of implicit-deadline tasks and let $\mathcal{S}_1 \dot\cup \ldots \dot\cup \mathcal{S}_k = \mathcal{S}$ be a $(1+\delta)$-feasible solution, i.e. each $\mathcal{S}_j$ is $(1+\delta)$-feasible. Let $\mathcal{B}_1, \ldots, \mathcal{B}_m$ be a block partition of $\mathcal{S}$ w.r.t. $\varepsilon$ and let $I_1, \ldots, I_\mu \subseteq [1,m]$ be disjoint intervals such that $\sum_{j \in I_i} u(\mathcal{B}_j) \leq \varepsilon$ for all $i \in \{1, \ldots, \mu\}$. For any interval $I_i$ we may reset the periods of the tasks arbitrarily within the period bounds of $\bigcup_{j \in I_i} \mathcal{B}_j$. Let $\mathcal{S}'_1 \dot\cup \ldots \dot\cup \mathcal{S}'_k = \mathcal{S}'$ be the solution with the modified tasks. Then each $\mathcal{S}'_j$ is $(1+\delta+\varepsilon)(1+2\varepsilon)$-feasible.*

*Proof.* To simplify notation, we can join blocks, such that $|I_i \cap \{1, \ldots, m\}| = 1$ for all intervals (this destroys the property that $\frac{p(\tau)}{p(\tau')}$ is upper bounded by a constant for all tasks $\tau, \tau'$ in the same block, but we do not need this property here). Consider w.l.o.g. $\mathcal{S}_1$ and some task $\tau \in \mathcal{S}_1$. Suppose $\tau \in \mathcal{B}_i$ and let $r \in [p(\tau)/2, p(\tau)]$ such that $D_{\tau,\mathcal{S}_1}(r) \leq (1+\delta)r$. Inspect the local response time function $D_{\tau,\mathcal{S}_1}^{\mathcal{B}_1,\ldots,\mathcal{B}_m}(r)$:

$$c(\tau) + r \cdot u((\mathcal{B}_1 \cup \ldots \cup \mathcal{B}_{i-1}) \cap \mathcal{S}_1) + r \cdot \sum_{\tau' \prec \tau : \tau' \in \mathcal{B}_i \cap \mathcal{S}_1} \underbrace{\left\lceil \frac{r}{p(\tau')} \right\rceil \frac{p(\tau')}{r}}_{\in [1,2]} \cdot u(\tau') \leq (1+\delta) r$$

Here one has $\lceil \frac{r}{p(\tau')} \rceil \frac{p(\tau')}{r} \leq 2$ since $\frac{r}{p(\tau')} \geq \frac{r}{p(\tau)} \geq 1/2$. First suppose that $\tau$ does not lie in one of the blocks, whose index is in an interval. Then resetting periods in any of the blocks $\mathcal{B}_1, \ldots, \mathcal{B}_{i-1}$ does not affect this function. Thus $\tau$ is still locally $(1+\delta)$-feasible and therefore $(1+\delta)(1+2\varepsilon)$-feasible by Lemma 3.5. Now suppose $\tau \in \mathcal{B}_i$ and $i \in I_{i'}$ for some $i'$. Let $\mathcal{S}'_1, \ldots, \mathcal{S}'_k$ be the family of solutions $\mathcal{S}_1, \ldots, \mathcal{S}_k$ after resetting the periods. Similar $\mathcal{B}'_1, \ldots, \mathcal{B}'_m$ emerge from $\mathcal{B}_1, \ldots, \mathcal{B}_m$. By choosing $r := p(\tau)$, we can upper bound

[1] We may allow ±∞ as borders for the first and last interval, resp.

$D^{\varepsilon}_{\tau,\mathcal{S}'_1}(p(\tau))/p(\tau)$ by

$$\begin{aligned}
&\frac{c(\tau)}{p(\tau)} + u((\mathcal{B}'_1 \cup \ldots \cup \mathcal{B}'_{i-1}) \cap \mathcal{S}'_1) + \sum_{\tau' \in \mathcal{B}'_i \cap \mathcal{S}'_1 : \tau' \prec \tau} \underbrace{\left\lceil \frac{p(\tau)}{p(\tau')} \right\rceil \frac{p(\tau')}{p(\tau)}}_{\leq 2} \cdot u(\tau') \\
\leq\ & u((\mathcal{B}_1 \cup \ldots \cup \mathcal{B}_{i-1}) \cap \mathcal{S}_1) + 2\underbrace{u(\mathcal{B}_i \cap \mathcal{S}_1)}_{\leq \varepsilon} \\
\leq\ & 1 + \delta + \varepsilon
\end{aligned}$$

using that $u(\mathcal{B}_i \cap \mathcal{S}_1) \leq u(\mathcal{B}_i) \leq \varepsilon$ and that we did not change the utilization of any task. Again by Lemma 3.5 the task system $\mathcal{S}'_1$ is $(1 + \delta + \varepsilon)(1 + 2\varepsilon)$-feasible. □

We still have to show that we did not discard too many tasks during the clustering.

**Lemma 3.7.** *Let* $\mathcal{S}^d \subseteq \mathcal{S}$ *be the set of tasks, discarded in the course of the clustering procedure and* $0 < \varepsilon \leq 1/2$*. One has* $u(\mathcal{S}^d) \leq 5\varepsilon \cdot u(\mathcal{S})$*.*

*Proof.* For obtaining the block partitions we discard tasks of utilization at most $\varepsilon \cdot u(\mathcal{S})$. Next consider the cases that some blocks $\mathcal{B}_i, \ldots, \mathcal{B}_{j-1}$ are discarded. This happens if $u(\mathcal{B}_i \cup \ldots \cup \mathcal{B}_{j-1}) \leq \varepsilon^2$, but $u(\mathcal{B}_i \cup \ldots \cup \mathcal{B}_j) > \varepsilon$. Consequently $u(\mathcal{B}_j) \geq \varepsilon - \varepsilon^2 \geq \varepsilon/2$. Now let us account the discarded utilization of at most $\varepsilon^2$ to the utilization of $\mathcal{B}_j$. We will never again charge $\mathcal{B}_j$. Clearly the accounted utilization is in total at most $\frac{\varepsilon^2}{\varepsilon/2} u(\mathcal{S}) = 2\varepsilon \cdot u(\mathcal{S})$. Note that just in the last step, we might have $u(\mathcal{B}_i \cup \ldots \cup \mathcal{B}_m) \leq \varepsilon^2$, but no block to account for discarding $\mathcal{B}_i, \ldots, \mathcal{B}_m$. Finally consider the tasks, which are discarded in step (5) in some block $\mathcal{B}_i$ due to the reason, that the tasks with periods $p$ have a to tiny utilization. We account the discarded utilization of $\varepsilon^6$ to $\mathcal{B}_i$. Since there are at most $1/\varepsilon^3$ many periods for small tasks per block, we account at most $\varepsilon^6 \cdot \frac{1}{\varepsilon^3} = \varepsilon^3$ to $\mathcal{B}_i$. Since $u(\mathcal{B}_i) \geq \varepsilon^2$ the total utilization of tasks, discarded in step (5) is bounded by $\varepsilon^3 u(\mathcal{S})/\varepsilon^2 \leq \varepsilon \cdot u(\mathcal{S})$.

Accumulating the bounds yields that in total the discarded utilization is at most

$$\varepsilon \cdot u(\mathcal{S}) + 2\varepsilon u(\mathcal{S}) + \varepsilon^2 + \varepsilon \cdot u(\mathcal{S}) \leq 5\varepsilon \cdot u(\mathcal{S})$$

since we assume that $u(\mathcal{S}) \geq 1/2 \geq \varepsilon$. □

## Merging

The situation after the above clustering step is now as follows: For a constant $\gamma = \varepsilon^6$, the amount of utilization of small tasks $\mathcal{S}_s = \{\tau \in \mathcal{S} \mid u(\tau) \leq \gamma\}$ of a period is either 0 or at least $\gamma$. Thus we can partition the small tasks $\mathcal{S}_s$ into *groups* $R_1, \ldots, R_q$ such that for all groups $R_i$ one has

- $p(\tau) = p(\tau') \quad \forall \tau, \tau' \in R_i$
- $\gamma \leq u(R_i) \leq 3\gamma$

The merging procedure now consists of replacing the tasks in a group $R_i$ by a single task $\tau_i$, such that $p(\tau_i)$ equals the periods of the tasks in $R_i$ and $u(\tau_i) = u(R_i)$. In other words, we *glue* small tasks of the same period together. Let $\mathcal{S}'$ be the emerging task system, now consisting only of large tasks.

It is clear that any solution to $\mathcal{S}'$ immediately gives a feasible solution of the same cost for $\mathcal{S}$, but it is less obvious, whether we can still guarantee solutions for $\mathcal{S}'$, which are not significantly more costly than $OPT_{\textsc{MulSched}}(\mathcal{S})$, since we loose some flexibility to distribute small tasks. Those belonging to one group must be assigned to the same processor.

In the proof of the next theorem we will apply a non-standard variant of the Chernoff bound. Recall that given a sum $X := X_1 + \ldots + X_n$ of independently distributed random variables $X_i \in \{0, 1\}$, it is well known that $X$ is distributed sharply around its mean. More precisely for each $\mu \geq E[X]$ one has

$$\Pr[X \geq (1 + \delta)\mu] \leq e^{-\delta^2 \mu/3}$$

for $0 < \delta < 1$, see e.g. the book of Mitzenmacher & Upfal [MU05]. A similar result holds for weighted sums, if the weights do not differ too much.

**Theorem 3.8.** *Let $X_1, \ldots, X_n$ be independent random variables with $X_i \in \{0, b_i\}$ for $b_i > 0$. Consider the sum $X := a_1 X_1 + \ldots + a_n X_n$ with $a_i > 0$. Then for $\alpha := \max_i\{a_i \cdot b_i\}$ and $0 < \delta < 1$ one has*

$$\Pr[X \geq (1 + \delta)E[X]] \leq e^{-\frac{\delta^2}{3\alpha}E[X]}.$$

A proof can be found in the Appendix. We next show, that the merging step doesn't increase the value of $OPT$ significantly — at least if we again allow a slight processor speedup.

**Lemma 3.9.** *Let $\gamma = \varepsilon^6$, $0 \leq \delta \leq 1, 0 < \varepsilon \leq 1/3$ and let $\mathcal{S}$ be a set of tasks which can be partitioned into subsets $\mathcal{S}_\ell = \{\tau \in \mathcal{S} \mid u(\tau) > \gamma\}$ of large tasks and groups $R_1, \ldots, R_q$ with $\gamma \leq u(R_i) \leq 3\gamma$. Glue together tasks in $\mathcal{S}$ from the same group and denote the emerging task system by $\mathcal{S}'$. Let $\mathcal{S} = \mathcal{S}_1 \dot\cup \ldots \dot\cup \mathcal{S}_k$ be a $(1 + \delta)$-feasible solution for $\mathcal{S}$. Then there is a solution of $k$ processors for $\mathcal{S}'$ which is $(1 + \delta + 2\varepsilon)$-feasible, after discarding tasks of utilization $\varepsilon \cdot u(\mathcal{S})$.*

*Proof.* Let a $(1 + \delta)$-feasible solution $\mathcal{S}_1 \dot\cup \ldots \dot\cup \mathcal{S}_k = \mathcal{S}$ be given. We have to show that there exists a solution for $\mathcal{S}$ which uses at most $k$ processors of speed $1 + \delta + 2\varepsilon$, in which the tasks of each group $R_i$ are scheduled together on one processor. In the construction of this solution we are allowed to discard an $\varepsilon$-fraction of the tasks.

After identifying sets $\mathcal{S}_i \cap R_j$ with a single task of utilization $u(\mathcal{S}_i \cap R_j)$ we may assume that each processor contains at most one task from each group $R_j$. The new solution $\mathcal{S}'_1, \ldots, \mathcal{S}'_k$ is now constructed using a probabilistic argument. We first assign deterministically all large tasks from $\mathcal{S}_i$ to $\mathcal{S}'_i$. Furthermore all tasks from group $R_j$ are assigned to $\mathcal{S}'_i$ with probability $\frac{u(R_j \cap \mathcal{S}_i)}{u(R_j)}$.

We consider the situation on processor $i$. First suppose that $\tau \in \mathcal{S}_i$ is a large task, i.e. $\tau \in \mathcal{S}_\ell$. We need to show that after moving small tasks, $\tau$ still meets the deadlines, at least after a slight speedup of the processor. Let $r \in [p(\tau)/2, p(\tau)]$ such that $D_{\tau,\mathcal{S}_i}(r) \leq (1+\delta)r$.

We show that $\Pr[D_{\tau,\mathcal{S}'_i}(r) > (1+\delta+2\varepsilon)r] \leq \varepsilon$, i.e. it is unlikely that the response time function grows significantly. After reordering we may assume that $\tau_1,\dots,\tau_m$ are the small tasks in $\mathcal{S}_i$ of higher priority than $\tau$ and $R_j \cap \mathcal{S}_i = \{\tau_j\}$. Then $D_{\tau,\mathcal{S}_i}(r)$ can be rewritten as

$$c(\tau) + \sum_{\tau' \in \mathcal{S}_i \cap \mathcal{S}_\ell, \tau' \prec \tau} \left\lceil \frac{r}{p(\tau')} \right\rceil \cdot c(\tau') + r \cdot \sum_{j=1}^{m} \underbrace{\left\lceil \frac{r}{p(\tau_j)} \right\rceil \frac{p(\tau_j)}{r}}_{=:a_j \in [1,2]} \cdot u(\tau_j).$$

Note that the number $a_j = \lceil \frac{r}{p(\tau_j)} \rceil \cdot \frac{p(\tau_j)}{r}$ in the right sum satisfies $1 \leq a_j \leq 2$. After randomly redistributing the tasks in $R_1,\dots,R_q$, the evaluation of the response time function at $r$ is a random variable of the form

$$D_{\tau,\mathcal{S}'_i}(r) = c(\tau) + \sum_{\tau' \in \mathcal{S}_i \cap \mathcal{S}_\ell, \tau' \prec \tau} \left\lceil \frac{r}{p(\tau')} \right\rceil \cdot c(\tau') + r \cdot \sum_{j=1}^{m} a_j \cdot X_j$$

where the $X_i \in \{0, u(R_j)\}$ are independent random variables with $\Pr[X_j = u(R_j)] = \frac{u(\tau_j)}{u(R_j)}$. For $X := \sum_{j=1}^m a_j X_j$, one has $E[X] \leq 1+\delta \leq 2$. It is sufficient to show that $\Pr[X \geq E[X]+\varepsilon] \leq \varepsilon$. Here the variant of the Chernoff bound (Theorem 3.8) comes into play. Choose

$$\alpha := \max_j \{a_j \cdot u(R_j)\} \leq 2 \cdot 3\varepsilon^6 = 6\varepsilon^6.$$

Applying Theorem 3.8 yields, that

$$\Pr[X \geq E[X]+\varepsilon] = \Pr\left[X \geq \left(1 + \frac{\varepsilon}{E[X]}\right) E[X]\right] \leq e^{-\frac{1}{6\varepsilon^6} \frac{\varepsilon^2}{3E[X]^2} E[X]} \leq \varepsilon,$$

where the last inequality follows from $E[X] \leq 2$ and $\varepsilon \leq 1/3$.

Now consider the case that $u(\tau) \leq \varepsilon^6$, that means $\tau$ is a small task and $c(\tau)$ in (3.3) is a random variable as well. But then the above analysis can be applied after the observation that $u(\tau)$ grows up to at most $3\gamma$. Thus after the merging step, $c(\tau)$ is bounded by $3\gamma \cdot p(\tau) \leq 6\gamma \cdot r \leq \varepsilon \cdot r$. We conclude: With probability of at least $1-\varepsilon$, a given task $\tau$ is $(1+\delta+2\varepsilon)$-feasible. In the unlikely event that $\tau$ becomes infeasible, we discard $\tau$. The expected cumulated utilization of all discarded tasks is upper bounded by $\varepsilon \cdot u(\mathcal{S})$. By Lemma 3.2 and the principle of the probabilistic method (see e.g. the book of Alon & Spencer [AS08]) there is a way to distribute $\mathcal{S}$ among $k$ processors of speed $1+\delta+2\varepsilon$, such that tasks from each group $R_j$ are assigned to the same processor and we only need to discard tasks with a utilization of at most $\varepsilon \cdot u(\mathcal{S})$. □

## Post-processing

Suppose we obtain a MulSched solution for the task system, as modified in the clustering and merging step. Then this solution might be infeasible for the original tasks, because we changed the periods of some tasks. Since we did not modify tasks with a utilization of $\varepsilon$ or larger, this infeasibility must be due to small tasks. We next show that one can always remove a few small tasks to achieve again RM-schedulability. More formal

**Lemma 3.10.** *Let $0 \leq \delta \leq 1$ and $0 < \varepsilon \leq 1/2$. Given an implicit-deadline task system $\mathcal{S}$ let $\mathcal{S} = \mathcal{S}_\ell \cup \mathcal{S}_s$ be the partition into large tasks $\mathcal{S}_\ell$ with utilization larger than $\varepsilon$ and small tasks $\mathcal{S}_s$. If $\mathcal{S}$ is $(1+\delta)$-feasible and $\mathcal{S}_\ell$ is feasible, then one can discard a subset $\mathcal{S}' \subseteq \mathcal{S}_s$ of the small tasks with a utilization of $u(\mathcal{S}') \leq \delta + \varepsilon$, such that $\mathcal{S} \backslash \mathcal{S}'$ is RM-schedulable.*

*Proof.* Let $\{\tau_1, \ldots, \tau_k\} = \mathcal{S}_s$ be the small tasks, ordered by their periods, i.e. $\tau_1 \prec \ldots \prec \tau_k$. If $u(\mathcal{S}_s) < \delta + \varepsilon$, discarding all small tasks, makes $\mathcal{S}$ feasible, thus assume that $u(\mathcal{S}_s) \geq \delta + \varepsilon$. Choose a $j \in \{1, \ldots, k\}$ such that $\delta \leq u(\tau_1) + \ldots + u(\tau_j) \leq \delta + \varepsilon$. Define $\mathcal{S}' := \{\tau_1, \ldots, \tau_j\}$. It remains to show feasibility of an arbitrary task $\tau \in \mathcal{S} \backslash \mathcal{S}'$. If $\tau \prec \tau_j$, then $\tau$ is trivially feasible, since all higher priority tasks and $\tau$ itself must be contained in $\mathcal{S}_\ell$ and $\mathcal{S}_\ell$ is feasible by assumption. Thus assume $\tau_j \prec \tau$. Since $\mathcal{S}$ is $(1+\delta)$-feasible, we know that there is an $r \in [p(\tau)/2, p(\tau)]$ with

$$D_{\tau,\mathcal{S}}(r) = c(\tau) + r \cdot \sum_{\tau' \in \mathcal{S} : \tau' \prec \tau} \underbrace{\left\lceil \frac{r}{p(\tau')} \right\rceil \frac{p(\tau')}{r}}_{\geq 1} \cdot u(\tau') \leq (1+\delta) r$$

whereby we rewrote the response time function in a suitable way. Now discarding all tasks $\tau_1, \ldots, \tau_j$ decreases the left hand side by at least $r \cdot u(\mathcal{S}') \geq r \cdot \delta$. Thus $\mathcal{S} \backslash \mathcal{S}'$ is RM-schedulable. □

We now have everything by hand to derive the main result of this chapter

**Theorem 3.11** (Merging Theorem). *Let $0 < \varepsilon \leq 1/3$. Let $\mathcal{S}$ be any implicit-deadline task system. Then using the clustering, merging and preprocessing steps, described above, we obtain in polynomial time a weakly $\mathcal{O}(\varepsilon)$-dominating set $\mathcal{S}'$ with $u(\tau) \geq \gamma = \varepsilon^6$ for all $\tau \in \mathcal{S}'$ and*

$$OPT_{\text{MulSched}}(\mathcal{S}') \leq (1 + \mathcal{O}(\varepsilon)) \cdot OPT_{\text{MulSched}}(\mathcal{S}) + \mathcal{O}(1).$$

*Proof.* It suffices to show the following two assertions.

Claim: *One has $OPT_{\text{MulSched}}(\mathcal{S}') \leq (1 + \mathcal{O}(\varepsilon)) \cdot OPT_{\text{MulSched}}(\mathcal{S}) + \mathcal{O}(1)$.*
Let $\mathcal{S} = \mathcal{S}_1 \cup \ldots \cup \mathcal{S}_k$ be any feasible solution for the original task system. We applied the following operations to obtain a modified task system: Rounding periods, creating blocks, clustering tasks, merging tasks. In each operation the sets $\mathcal{S}_1, \ldots, \mathcal{S}_k$ remain feasible after a speedup of $1 + \mathcal{O}(\varepsilon)$ and discarding an $\mathcal{O}(\varepsilon)$-fraction of the tasks, see Lemmas 3.6, 3.7 and 3.9. We obtain the existence of a partition $\mathcal{S}' = \mathcal{S}'_1 \dot{\cup} \ldots \dot{\cup} \mathcal{S}'_{k'}$ with $k' = (1 + \mathcal{O}(\varepsilon))k + \mathcal{O}(1)$ for the modified task system, such that each $\mathcal{S}'_i$ is $(1 + \mathcal{O}(\varepsilon))$-feasible. Since the modifications affect only tasks with utilization less than $\varepsilon$, $\mathcal{S}'_i \cap \{\tau \in \mathcal{S}' \mid u(\tau) \geq \varepsilon\}$ is RM-schedulable. Lemma 3.10 then yields that tasks of utilization $\mathcal{O}(\varepsilon)$ can be removed to make $\mathcal{S}'_i$ feasible. The total amount discarded utilization is then $\mathcal{O}(\varepsilon) \cdot k$.

Claim: *$\mathcal{S}'$ is weakly $\mathcal{O}(\varepsilon)$-dominating $\mathcal{S}$.*
Now we go the other way around. Let $\mathcal{S}' = \mathcal{S}'_1 \dot{\cup} \ldots \dot{\cup} \mathcal{S}'_k$ be any feasible solution for $\mathcal{S}'$. Let $\mathcal{S}_i$ be the tasks in $\mathcal{S}$, corresponding to that in $\mathcal{S}'_i$. We have to show that we can make $\mathcal{S}_i$ feasible, after discarding an $\mathcal{O}(\varepsilon)$-fraction of the tasks. The only reason, why $\mathcal{S}_i$ might be infeasible is due to the resetting of periods. However, by applying Lemma 3.6 again, $\mathcal{S}_i$ is at least $(1+\varepsilon)(1+2\varepsilon) \leq 1 + 4\varepsilon$ feasible. Since $\mathcal{S}_i$ is feasible if restricted to

tasks of utilization at least $\varepsilon$, Lemma 3.10 makes $\mathcal{S}_i$ feasible after discarding task with a cumulated utilization of at most $5\varepsilon$. It follows from the proof in Lemma 3.10, that these tasks can be found in polynomial time. Again the total amount of discarded tasks is clearly $\mathcal{O}(\varepsilon) \cdot k$, implying the claim. □

A consequence of this theorem is the following powerful claim

**Corollary 3.12.** *Suppose there is an algorithm $A_\varepsilon$ which for any fixed $\varepsilon > 0$ computes in polynomial time a solution of cost $(1+\varepsilon)OPT + \mathcal{O}(1)$ for* MulSched *instances $\mathcal{S}$ with $u(\tau) \geq \varepsilon\ \forall \tau \in \mathcal{S}$. Then for any $\varepsilon > 0$, there is a polynomial time algorithm $A'_\varepsilon$ with*

$$A'_\varepsilon(\mathcal{S}) \leq (1+\varepsilon)OPT(\mathcal{S}) + \mathcal{O}(1)$$

*for all implicit-deadline task systems $\mathcal{S}$.*

## 3.4 The algorithm

Still, let $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$ be an implicit-deadline task system. Due to the work in the last sections, we now may assume that $u(\tau) \geq \gamma = \varepsilon^6$ for all task $\tau \in \mathcal{S}$ and an arbitrary but fixed $\varepsilon > 0$. Unfortunately, this assumption does not suffice for our algorithmic approach, which we are going to describe now. We will also need to modify large tasks, which we did not yet do so far. This might cause that the obtained solutions are slightly infeasible for the original instance.

### Rounding the instance

We suggest the following rounding procedure, which is now applied to all tasks

- Round up the periods $p(\tau)$ of all tasks to the next larger power of $1+\varepsilon$.
- Round down the running time $c(\tau)$ of any task $\tau$, such that the utilization $u(\tau)$ becomes an integer multiple of $\gamma^2 = \varepsilon^{12}$.
- Create (similar to Section 3.3) a block partition $\mathcal{S} \backslash \mathcal{S}^d = \mathcal{B}_1 \dot\cup \ldots \dot\cup \mathcal{B}_m$, after discarding $\mathcal{S}^d \subseteq \mathcal{S}$ with $u(\mathcal{S}^d) \leq \varepsilon \cdot u(\mathcal{S})$, i.e. $p(\tau) \leq \varepsilon \cdot p(\tau')$ for $\tau \in \mathcal{B}_i, \tau' \in \mathcal{B}_j$ with $i < j$ and the tasks in each block have at most $1/\varepsilon^3$ many different periods.

The rounding is visualized in Figure 3.2. Let $\mathcal{S}'$ be the obtained *well rounded* task system, then clearly $OPT_{\text{MulSched}}(\mathcal{S}') \leq OPT_{\text{MulSched}}(\mathcal{S})$, since the running times were rounded down and the periods have been rounded up. On the other hand let $\mathcal{S}'_i \subseteq \mathcal{S}'$ be a feasible task system and let $\mathcal{S}_i$ be the corresponding original tasks in $\mathcal{S}$. Then $\mathcal{S}_i$ is at least feasible on a processor of speed $(1+\varepsilon)^2 \leq 1 + 3\varepsilon$, since periods and running times are both scaled by not more than $1+\varepsilon$ (while this is clear for the periods, note that the ratio of old and new utilization is bounded by $\frac{u(\tau)}{u(\tau)-\gamma^2} \leq \frac{1}{1-\gamma} \leq 1 + 2\gamma \leq 1 + \varepsilon$).

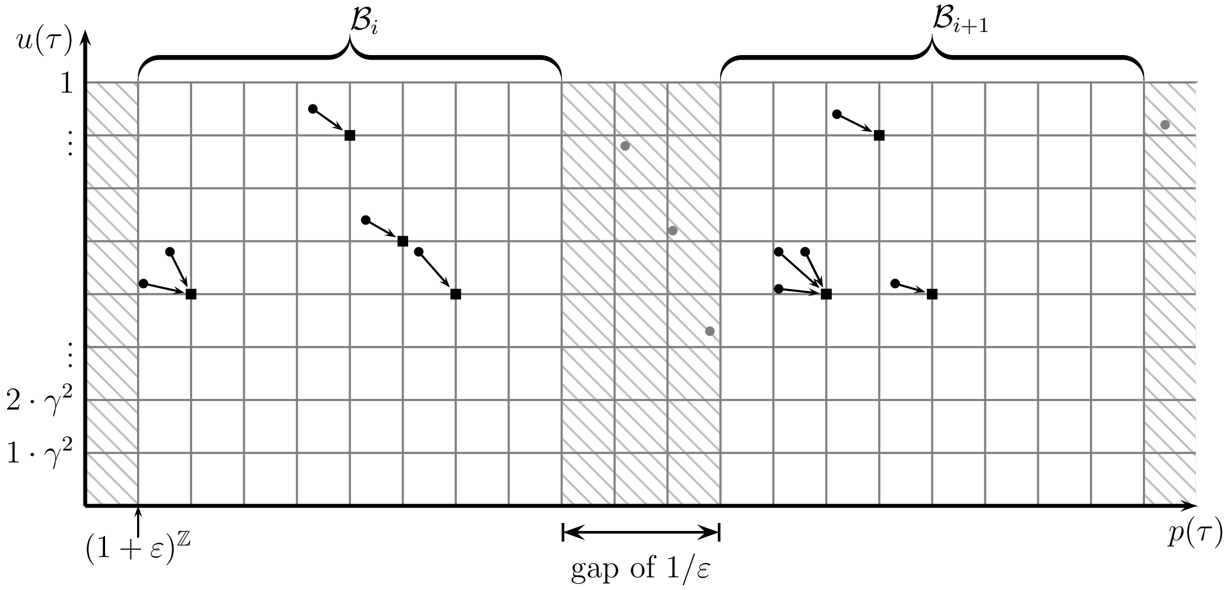


Figure 3.2: Rounding tasks to discretize instance. Balls indicate original tasks, rectangles denote rounded tasks. Gray balls are discarded tasks.

## A dynamic program

Recalling Lemma 3.4, we observe that it suffices to compute an optimum locally feasible solution.

**Theorem 3.13.** *For a well-rounded implicit-deadline task system $\mathcal{S}$, an optimum locally feasible solution (w.r.t. the block partition $\mathcal{B}_1, \dots, \mathcal{B}_m$) can be computed in time $\mathcal{O}_\varepsilon(1) \cdot n^{(1/\varepsilon)^{\mathcal{O}(1)}}$ with $n = |\mathcal{S}|$.*

For the rest of this chapter, we use the term "locally feasible" always w.r.t. the block partition $\mathcal{B}_1, \dots, \mathcal{B}_m$. To obtain the above theorem, we demonstrate how the tasks can be distributed with a dynamic programming algorithm to compute an optimal assignment of $\mathcal{S}$, such that each task is locally feasible. This is done block-wise via dynamic programming. The key ingredients, to make our algorithm work in polynomial time, are:

1. The number of different task types per block is bounded by a constant.
2. In the concept of local feasibility, the tasks from blocks $\mathcal{B}_1, \dots, \mathcal{B}_{\ell-1}$ influence the feasibility of tasks in $\mathcal{B}_\ell$ only with their utilization.

A vector $a = (a_0, \dots, a_{1/\gamma^2})$ with $a_i \in \mathbb{Z}$ is called a *configuration*, whereby $a_i$ denotes the number of processors whose utilization is exactly $i \cdot \gamma^2$. We require that $\sum_{i=0}^{1/\gamma^2} a_i = n$, since $|\mathcal{S}| = n$ many processors suffice in any case. Consider the following table entries.

$$A(a, \ell) \quad = \quad \begin{cases} 1 & \text{if tasks in } \mathcal{B}_1, \dots, \mathcal{B}_\ell \text{ can be scheduled in a locally feasible way} \\ & \text{such that the utilization bounds of configuration } a \text{ are exactly met} \\ 0 & \text{otherwise} \end{cases}$$

Note that $a$ has fixed dimension, thus the table has a polynomial number of entries. We now describe, how to compute $A(a, \ell)$ efficiently. Let $b = (b_0, \dots, b_{1/\gamma^2})$ be a processor configuration from a distribution of the tasks $\mathcal{B}_1, \ldots, \mathcal{B}_{\ell-1}$. Then $d(\mathcal{B}_\ell, b, a)$ is defined to be 1, if the tasks in block $\mathcal{B}_\ell$ can be additionally distributed among the processors, such that the bounds of configuration $a$ are met. The base case is

$$A(a, 1) = d(\mathcal{B}_1, (n, 0, \ldots, 0), a)$$

The configuration $(n, 0, \ldots, 0)$ here means, that initially all $n$ processors are empty. For all $\ell > 1$ note that $A(a, \ell) = 1$, if and only if there exists a configuration $b \in \mathbb{Z}^{1/\gamma^2+1}$ with $0 \leq b_i \leq a_i$ for all $i$ and

$$A(b, \ell - 1) = 1 \quad \text{and} \quad d(\mathcal{B}_\ell, b, a) = 1$$

In other words $A(a, \ell) = 1$ holds, if one can distribute the tasks from the first $\ell - 1$ blocks to reach a configuration $b$ and then add tasks in $\mathcal{B}_\ell$ in a feasible way, to extend this configuration to obtain $a$. After computing all entries, the optimal number of processors can be read out of the table. In fact, it is

$$\min\{k \in \mathbb{N}_0 \mid \exists \text{ configuration } a \text{ with } A(a, m) = 1 \text{ and } a_0 = n - k\}.$$

The concrete solution, attaining this value can be easily reconstructed.

## Computing $d(\mathcal{B}_\ell, b, a)$

It remains to show, how to determine $d(\mathcal{B}_\ell, b, a)$ in time $\mathcal{O}_\varepsilon(1) \cdot n^{\mathcal{O}(1)}$. Each block $\mathcal{B}_\ell$ has only a constant number of different task-types, each having a utilization, which is lower-bounded by a constant. Suppose that $\mathcal{B}_\ell$ has tasks, whose running-time and period are from the tuples $(c_1, p_1), \ldots, (c_k, p_k)$ with $k \leq \frac{1}{\gamma^2} \cdot \frac{1}{\varepsilon^3} = \mathcal{O}_\varepsilon(1)$. Let $d_i$ be the number of times, that task $(c_i, p_i)$ is contained in $\mathcal{B}_\ell$ and suppose that the tasks are ordered by their priorities, i.e. $p_1 \leq \ldots \leq p_k$. A *pattern* is a vector $(x_1, \ldots, x_k) \in \mathbb{N}_0^k$, which represents a set, composed of these task types (the set, defined by the pattern, contains $x_i$ times task type $(c_i, p_i)$). There is only a constant number of candidate patterns, which can be used to distribute the tasks in $\mathcal{B}_\ell$.

We are now going to show, how $d(\mathcal{B}_\ell, b, a)$ can be computed for configurations $a, b \in \mathbb{Z}^{1/\gamma^2+1}$. Let $\mathcal{P}_{\beta\alpha}$ be the set of all patterns $x$, such that $x$ can be assigned to a processor, having already tasks from blocks $\mathcal{B}_1, \ldots, \mathcal{B}_{\ell-1}$ with utilization $\beta \cdot \gamma^2$ without violating local feasibility and with increasing the total utilization of this processor to precisely $\alpha \cdot \gamma^2$ ($\alpha, \beta \in \mathbb{Z}$). More concrete, $\mathcal{P}_{\beta\alpha}$ consists of all $x$ such that

$$\begin{aligned} \exists 0 \leq r_j \leq p_j : x_j c_j + r_j \cdot \beta\gamma^2 + \sum_{i=1}^{j-1} \left\lceil \frac{r_j}{p_i} \right\rceil x_i c_i &\leq r_j \quad \forall j = 1, \ldots, k \\ \beta\gamma^2 + \sum_{i=1}^{k} x_i \frac{c_i}{p_i} &= \alpha\gamma^2 \end{aligned}$$

This conditions can be checked efficiently in time $\mathcal{O}_\varepsilon(1) \cdot n^{\mathcal{O}(1)}$ (for each $x$) via integer programming in fixed dimension [Len83].

Each pattern may occur in several $\mathcal{P}_{\beta\alpha}$'s, thus we ideally think of several copies of $x$, each assigned to precisely one $\mathcal{P}_{\beta\alpha}$. Due to the lower bound on the utilization, the number of tasks per pattern cannot exceed $1/\gamma$, hence the number of feasible patterns is again a (huge) constant.

Let $\lambda_x$ be a variable, indicating the number of times, that pattern $x$ is used, to distribute tasks $\mathcal{B}_\ell$. Then we have $d(\mathcal{B}_\ell, b, a) = 1$ if and only if the following system has a solution

$$\begin{aligned} \sum_x \lambda_x x &= d && \\ \sum_i \sum_{x \in \mathcal{P}_{ij}} \lambda_x &= a_j && \forall j \\ \sum_j \sum_{x \in \mathcal{P}_{ij}} \lambda_x &= b_i && \forall i \\ \lambda_x &\in \mathbb{Z}_{\geq 0} && \forall x \end{aligned}$$

Again we use [Len83] to solve this integer program with a fixed number of variables in time $\mathcal{O}_\varepsilon(1) \cdot n^{\mathcal{O}(1)}$. The final conclusion of this chapter is then

**Theorem 3.14.** *Let $0 < \varepsilon \leq 1/3$ with $\frac{1}{\varepsilon} \in \mathbb{Z}$ arbitrary, but fixed. Then there is an algorithm, which partitions an implicit-deadline task system $\mathcal{S}$ into $\mathcal{S}_1, \dots, \mathcal{S}_k$ with $k \leq (1+\varepsilon)OPT_{\text{MulSched}}(\mathcal{S}) + \mathcal{O}(1)$ in time $\mathcal{O}_\varepsilon(1) \cdot n^{(1/\varepsilon)^{\mathcal{O}(1)}}$, such that each $\mathcal{S}_i$ is $(1+\varepsilon)$-feasible.*

# Chapter 4

# An Optimum Algorithm in the Average

As a rule of thumb, a good heuristic for an optimization problem does not need to yield provably good solutions even in the worst case, but must be asymptotically optimal in the average, meaning if the input is drawn from a reasonable probability distribution. *Asymptotically optimal* here means that the expected approximation ratio must tend to 1 for growing instances. The reason is that for heuristics, that are optimal in the average, one can also hope for near optimal solutions for (large) instances appearing in practice.

In this chapter, we introduce a simple First Fit based algorithm, called *First Fit Matching Periods* (`FFMP`) and show that it has exactly this property, where the input is a set $\mathcal{S} = \{\tau_1, \dots, \tau_n\}$ of implicit-deadline tasks and we aim at finding a partition of $\mathcal{S}$ into $P_1, \dots, P_k$ such that each $P_i$ is RM-schedulable and $k$ is minimized. Our algorithm initially sorts the tasks according to their $\alpha$-values and then distributes them in a First Fit manner, using the sufficient feasibility criterion of Burchard et al. [BLOS95]. Recall that for

$$\alpha(\tau) = \log p(\tau) - \lfloor \log p(\tau) \rfloor \quad \text{and} \quad \beta(\mathcal{S}) = \max_{\tau \in \mathcal{S}} \alpha(\tau) - \min_{\tau \in \mathcal{S}} \alpha(\tau)$$

a sufficient RM-schedulability criterion for a set of tasks $\mathcal{S}$ is that

$$u(\mathcal{S}) \leq 1 - \beta(\mathcal{S}) \ln(2),$$

see Section 2.2 or [BLOS95]. A formal description of our algorithm is as follows

**Algorithm 1** `FFMP`

```
Input: Set $\mathcal{S} = \{\tau_1, \dots, \tau_n\}$ of implicit-deadline tasks
Output: Assignment of $\tau_1, \dots, \tau_n$ to processors $P_1, P_2, \dots$

(1) Sort tasks such that $0 \leq \alpha(\tau_1) \leq \alpha(\tau_2) \leq \dots \leq \alpha(\tau_n) < 1$
(2) FOR $i = 1, \dots, n$ DO
        (3) Assign $\tau_i$ to the processor $P_j$ with the least index $j$ such that
            $u(P_j \cup \{\tau_i\}) \leq 1 - \beta(P_j \cup \{\tau_i\}) \cdot \ln(2)$
```

It is not clear, what exactly one understands under a reasonable probability distribution. However, here we decide to generate tasks as follows

1. First arbitrary periods are given by an adversary.
2. Then for each task $\tau$, the utilization $u(\tau)$ is drawn uniformly and independently from $[0, 1]$.

We will measure the quality of a solution in terms of the *waste*, which is the ratio of idles times, cumulated over all processors, therefore a solution for task set $\mathcal{S}$, consisting of $k$ processors has a waste of $k - u(\mathcal{S})$. Clearly minimizing the waste is equivalent to minimization of the number of partitions.

The main result in this chapter is that FFMP produces a solution with expected waste of $\mathcal{O}(n^{3/4}(\log n)^{3/8})$ if $n$ tasks are drawn according to the above probability distribution. Consequently the expected approximation ratio of $1 + \mathcal{O}(n^{-1/4}(\log n)^{3/8})$ tends to 1 for $n \to \infty$, thus the solution is asymptotically optimal on average.

To the best of our knowledge, this is the first proof that any algorithm for MulSched admits this property w.r.t. a reasonable probability distribution.

The only probabilistic analysis so far for Rate-monotonic scheduling of implicit-deadline tasks deals with the single processor case. The result of Lehoczky, Sha & Ding [LSD89] indicates that the reachable processor utilization on average is much better, than the worst-case value of $\ln(2) \approx 69\%$. For example, if periods are drawn from $[1, 100]$ and the running times are scaled by the largest value, such that the system is barely schedulable, then the utilization tends to 88% for $n \to \infty$.

To achieve our results, we use the following approach: We introduce an auxiliary algorithm FFMP$^*$ and prove that for any task set, it needs at least as many processors as FFMP. Thus it suffices to derive an upper bound on the waste of this easier algorithm. We then point out that for suitable subsets of the input tasks, FFMP$^*$ behaves like a well studied BinPacking algorithm, termed FirstFit$^*$. Eventually this allows to bound the waste for FFMP$^*$ in terms of the waste of FirstFit$^*$.

Finally we discuss the worst-case behaviour of FFMP in Section 4.5 and show in Section 4.6 how it can be implemented to run in time $\mathcal{O}(n \log n)$ .

We begin by describing average-case analyses for BinPacking, focusing especially on the result of Shor [Sho84] on the waste of FirstFit and FirstFit$^*$.

## 4.1 Average case for Bin Packing

Recall that heuristics for BinPacking like FirstFit, NextFit, BestFit, as well as FirstFitDecreasing and BestFitDecreasing have in the worst case asymptotic approximation ratios between 11/9 and 2 [GGJY76, Joh73].

On the other hand, if the items are generated randomly, the heuristics perform much better, than in the worst-case scenarios. For item sizes drawn uniformly at random from $[0, 1]$, the BestFit algorithm yields an expected waste of $\Theta(\sqrt{n} \log^{3/4} n)$ [Sho84], while for FirstFit this value is lower bounded by $\Omega(n^{2/3})$ and upper bounded by $\mathcal{O}(n^{2/3}\sqrt{\log n})$ [Sho84]. In fact, a FirstFitDecreasing approach yields an even smaller waste of $\Theta(\sqrt{n})$ [Fre80, Knö81, Lue82]. NextFit on the other hand produces a linear waste of $(\frac{1}{3} \pm o(1))n = \Omega(n)$ [CSHY80]. If item sizes are drawn uniformly from $[0, \alpha]$, for any constant $\alpha \leq 1/2$, the waste of FirstFitDecreasing is even constant with high probability. Note that here the waste is defined similar to multiprocessor scheduling, namely as the number of bins minus the sum of all item sizes. See also the survey of Coffman, Garey & Johnson [CGJ84].

## 4.2 The result of Shor

We are especially interested in the result of Shor [Sho84], that the expected waste for `FirstFit` (w.r.t. item sizes uniformly distributed from $[0, 1]$) is upper bounded by $\mathcal{O}(n^{2/3}\sqrt{\log n})$, since we want to adapt it for `FFMP`. For the sake of completeness `FirstFit` is stated as Algorithm 2.

**Algorithm 2** `FirstFit`

```
Input: Items $I = \{a_1, \dots, a_n\}$ ($a_i \in [0, 1]$)
Output: Assignment of items to bins $B_1, B_2, \dots$
(1) FOR $i = 1, \dots, n$ DO
    (2) Assign item $a_i$ to the bin $B_j$ with the smallest index $j$
        such that size$(B_j) + a_i \leq 1$
```

Shor first introduced a simplified algorithm, termed `FirstFit`$^*$, which is `FirstFit` with the additional restriction, that an item is never assigned to a bin, owning already an item of size less than 1/2.

**Algorithm 3** `FirstFit`$^*$

```
Input: Items $I = \{a_1, \dots, a_n\}$ ($a_i \in [0, 1]$)
Output: Assignment of items to bins $B_1, B_2, \dots$
(1) FOR $i = 1, \dots, n$ DO
    (2) Assign item $a_i$ to the bin $B_j$ with the smallest index $j$ such that either $B_j$ is
        empty or both of the following conditions hold
        • $B_j$ contains one item and this item has size $\geq 1/2$
        • size$(B_j) + a_i \leq 1$
```

That means each bin in the solution contains either 1 or 2 items. Although it seems intuitive that restricting the choices of an approximation algorithm should not improve the quality of a solution, this is not true in general for all algorithms. But with some effort, one can prove that `FirstFit`$^*(I) \geq$ `FirstFit`$(I)$ for all instances $I$. Here, for a BinPacking instance $I$ and an algorithm `A`, by a slight abuse of notation `A(I)` denotes both, the solution and the value of the solution, which is produced if applying `A` to $I$.

In a first step, a simple case analysis yields that `FirstFit`$^*$ is monotonic, i.e.

$$\texttt{FirstFit}^*(I\backslash\{a_i\}) \leq \texttt{FirstFit}^*(I)$$

for all instances $I$ and items $a_i \in I$. Here, the simplicity of the algorithm is heavily exploited. Iterating this argument of course yields that removing any subset of items, can only lower the number of bins in the produced solution.

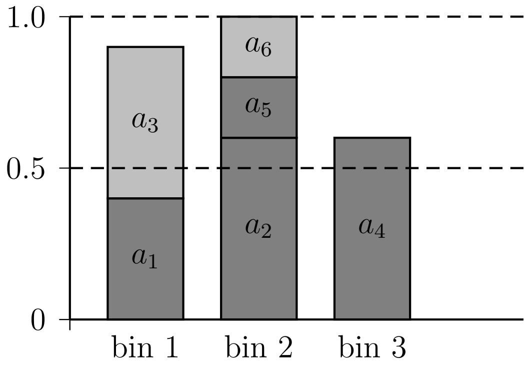


Figure 4.1: Example of an instance $I = (a_1, \dots, a_6) = (0.4, 0.6, 0.5, 0.6, 0.2, 0.2)$, assigned to bins by `FirstFit`. Dark items are also part of sub-instance $I' = (a_1, a_2, a_4, a_5)$. Observe that indeed $\texttt{FirstFit}^*(I') = \texttt{FirstFit}(I)$.

Next consider an instance $I$ and obtain a sub-instance $I' \subseteq I$ as follows: Remove all items from $I$ that are assigned by `FirstFit` to bins, which at that time contain at least one task of utilization less than 1/2 (see Figure 4.1 for an example). In other words, we obtain $I'$, by removing all items from $I$, which are assigned to bins, that $\texttt{FirstFit}^*$ would have considered to be full.

As a consequence, the restricted algorithm distributes $I'$ in the same way, that `FirstFit` distributes $I'$ in the solution $\texttt{FirstFit}(I)$. Hence

$$\texttt{FirstFit}(I) = \texttt{FirstFit}^*(I') \leq \texttt{FirstFit}^*(I)$$

yields the desired domination claim.

Now it suffices to give an upper bound on the expected waste of the simplified algorithm $\texttt{FirstFit}^*$. To this end, we transfer our analysis into a geometric setting. For each item $a_i$, we create a point $(a_i, i/n)$ in the unit square. If $a_i \leq 1/2$ we call it a $-$ point, otherwise a $+$ point. Next, mirror the $+$ points at the $x = 1/2$ line (see Figure 4.2 ($a$)).

We may now interpret the $\texttt{FirstFit}^*$ algorithm as follows: Match each $-$ point $a$ to that $+$ point $b$, which is to the right of $a$ and is most upwards. If such a $+$ point does not exist, leave the $-$ point unmatched (see Figure 4.2 ($b$)). Note that a $+$ point $p_+$ lies above a $-$ point $p_-$, if the large item corresponding to $p_+$ arrives before $p_-$. Furthermore $p_+$ lies to the right of $p_-$ if and only if the item sizes sum up to a value smaller than 1.

The expected waste can now be bounded by the expected number of unmatched points.

We introduce some notation for the further analysis. An *upright matching* is a matching between the points such that a $-$ point is matched to a $+$ point that is above and to the right. Additionally we call a matching *well-ordered*, if for pairs $(a, b), (c, d)$ of matched points ($p = (x_p, y_p) \in \mathbb{R}^2$ for $p \in \{a, b, c, d\}$) with $y_a > y_c > y_d > y_b$ one has $x_b < x_c$. In Figure 4.3 the left pair is well-ordered, the right one is not. We observe that $\texttt{FirstFit}^*$ produces well-ordered matchings. In fact, one can prove, that among all well-ordered upright matchings, the $\texttt{FirstFit}^*$ algorithm finds one of maximum cardinality. Thus it remains to show that there exists a well-ordered upright matching that leaves in expectation only $\mathcal{O}(n^{2/3}\sqrt{\log n})$ many nodes unmatched.

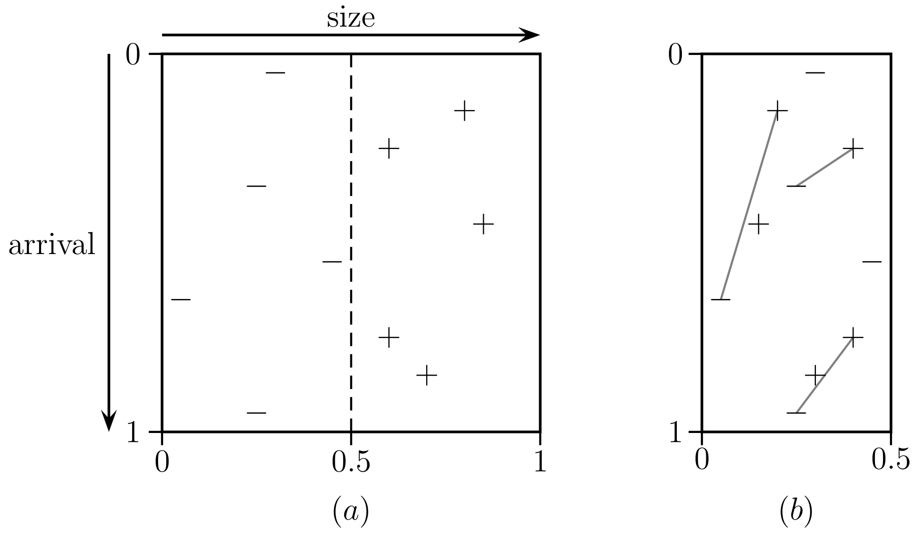


Figure 4.2: $(a)$ shows items turned into $-$ and $+$ points. In $(b)$ a $+$ point with coordinates $(x, y)$ is mapped to $(1 - x, y)$. Edges denote the upright matching, which is produced by `FirstFit`$^*$.

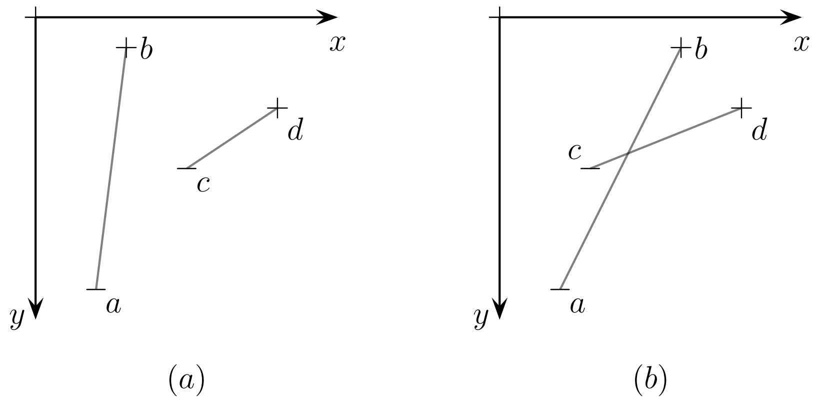


Figure 4.3: Pair of edges in $(a)$ is well-ordered, while this is not true for $(b)$.

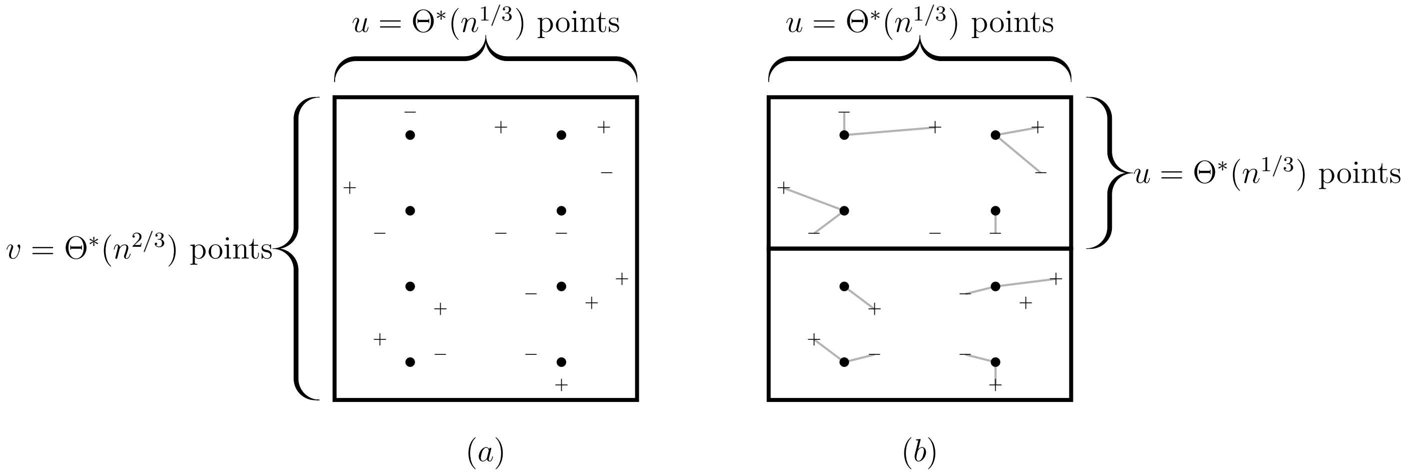

$u = \Theta^*(n^{1/3})$ points
$u = \Theta^*(n^{1/3})$ points
$v = \Theta^*(n^{2/3})$ points
$u = \Theta^*(n^{1/3})$ points
$(a)$
$(b)$

Figure 4.4: Unit square with $v \times u$ grid in $(a)$. Balls depict grid points. In $(b)$ rows are split into $v/u = \Theta^*(n^{1/3})$ many rectangles, each containing a $u \times u$ grid. Note that gray edges belong to $M_-$ or $M_+$, resp.

Here the following lemma is useful:

**Lemma 4.1** (Leighton, Shor [LS89])**.** *Draw $n$ points uniformly at random from $[0, 1]^2$. Then with probability of at least $1 - 1/n$, these points can be matched to grid points of a $\sqrt{n} \times \sqrt{n}$ grid such that all matching edges have length of $\mathcal{O}((\log n)^{3/4}/\sqrt{n})$.*

This lemma is not directly applicable to our setting, for example in our distribution the $y$-coordinates are not drawn uniformly at random. But given the point with the $j$th smallest $y$-coordinate w.r.t. uniform distribution, moving it to $y = j/n$ takes an expected distance of $\mathcal{O}(1/\sqrt{n})$. Next, transforming the half square $[0, 1] \times [0, 1/2]$ into the unit square can only affect constants. The same happens, if we assume that we have $n$ $-$ points and the same number of $+$ points. Thus we may at least assume that our distribution coincides with the assumptions from Lemma 4.1.

We now show, how to construct an upright matching, leaving $\mathcal{O}(n^{2/3}\sqrt{\log n})$ points unmatched. To simplify the presentation we use the $\mathcal{O}^*$-notation, which suppresses any poly-logarithmic term. This precision will be enough to explain the exponent of 2/3 — the poly-logarithmic term then drops out from calculus. First place a grid of $n = v \times u = \Theta^*(n^{2/3}) \times \Theta^*(n^{1/3})$ points in the unit square (see Figure 4.4 $(a)$). In other words, each row contains $u = \Theta^*(n^{1/3})$ many grid points, while each column contains $v = \Theta^*(n^{2/3})$ grid points.

We want to match the $+$ and $-$ points to the grid points. However, the grid is not quadratic in the sense, that a row would contain the same number of grid points than a column. But by grouping the rows, we may partition the grid into rectangles, each containing a $u \times u = \Theta^*(n^{1/3}) \times \Theta^*(n^{1/3})$ grid (see Figure 4.4 $(b)$). Each such rectangle contains in expectation $\Theta^*(n^{2/3})$ many $-$ and $+$ points. A simple application of the Chernoff bound (see e.g. [MU05, AS08]) yields that with high probability, the number of $-/+$ points differs by at most $\mathcal{O}^*(n^{1/3})$ from the number of grid points in the rectangle. Since we have $\mathcal{O}^*(n^{1/3})$ many squares, summing up the differences yields at most $\mathcal{O}^*(n^{2/3})$ points, which are discarded from now on.

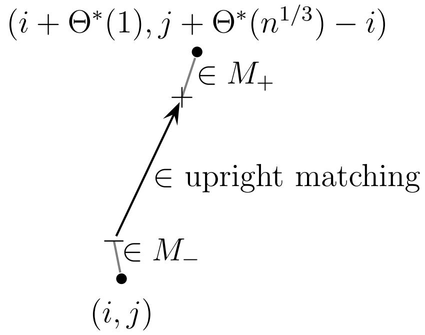


Figure 4.5: Construction of the upright matching out of $M_-, M_+$.

Though the rectangles are not unit squares, we can apply Lemma 4.1 to obtain with high probability matchings $M_-, M_+$, assigning the remaining + and − points to the grid points, such that edges in $M_-$ and $M_+$ cross at most $\mathcal{O}^*(1)$ grid lines (see Figure 4.4 ($b$)).

Next consider the − point, which is matched to grid point $(i,j)$ in $M_-$. We connect it to that + point, which is matched to the grid point $(i+\Theta^*(1), j+\Theta^*(n^{1/3})-i)$ in $M_+$ (see Figure 4.5).

We observe that this is in fact an upright matching. It leaves a few nodes uncovered, namely the − points in the highest $\mathcal{O}^*(n^{1/3})$ rows and in the rightmost $\mathcal{O}^*(1)$ columns (similar for the lowest/leftmost + points). However these are just $\mathcal{O}^*(n^{2/3})$ many.

As a technical difficulty, the obtained matching is not necessarily well-ordered, but by switching pairwise edges we may establish this property without decreasing the cardinality.

## 4.3 An auxiliary algorithm

We now begin to derive an upper on the expected waste for `FFMP`, if applied to a set of implicit-deadline tasks $\mathcal{S}$. Like `FirstFit`$^*$ is a restriction to `FirstFit`, we next state a restricted version of `FFMP`. Let $\gamma := \gamma(n)$ be an integer value, which we are going to choose later. First, the algorithm partitions the tasks into *groups* $\mathcal{S}_1, \ldots, \mathcal{S}_\gamma$ with $\mathcal{S}_j = \{\tau_i \in \mathcal{S} \mid \frac{j-1}{\gamma} \leq \alpha(\tau_i) < \frac{j}{\gamma}\}$, thus the $\alpha$-values of tasks from the same group differ only slightly. Next, `FFMP`$^*$ never assigns more than 2 tasks to each processor and tasks from different periods are never mixed. Here we say that an algorithm *mixes* two tasks $\tau_1, \tau_2$, if they are assigned to the same processor. The algorithm even considers a processor to be full, if the first assigned task has a utilization of at most $\approx 1/2$. Note that this algorithm is precisely tailored for the used probability distribution. A formal definition of `FFMP`$^*$ can be found as Algorithm 4.

Note that $1 - \frac{\ln(2)}{\gamma}$ is just slightly below 1. Observe that `FFMP`$^*$ assigns either one or two tasks to each processor. Let `FFMP`$^*(\mathcal{S})$ be the number of processors, needed when scheduling tasks $\mathcal{S}$ with algorithm `FFMP`$^*$. As a slight abuse of notation `FFMP`$^*(\mathcal{S})$ means as well the schedule, obtained when applying `FFMP`$^*$ to $\mathcal{S}$, however the meaning will be clear from the context. From Theorem 2.1 we see that the produced solution is always feasible, since either a single task is assigned to a processor or in case that two tasks are assigned, their $\alpha$-values differ by at most $1/\gamma$ and their cumulated utilization is upper bounded by $1 - \ln(2)/\gamma$.

**Algorithm 4** FFMP$^*$

**Input:** Set $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$ of implicit-deadline tasks
**Output:** Assignment of $\tau_1, \ldots, \tau_n$ to processors $P_1, P_2, \ldots$

```
(1) Sort tasks such that $0 \leq \alpha(\tau_1) \leq \ldots \leq \alpha(\tau_n) < 1$
(2) Partition tasks into groups $\mathcal{S}_1, \ldots, \mathcal{S}_\gamma$ with $\mathcal{S}_j = \{\tau_i \in \mathcal{S} \mid \frac{j-1}{\gamma} \leq \alpha(\tau_i) < \frac{j}{\gamma}\}$.
(3) FOR $i = 1, \ldots, n$ DO
        (4) Assign $\tau_i$ to the processor $P_j$ with the least index $j$ such that either $P_j$ is empty
            or all following conditions are satisfied
            (a) $P_j$ contains only one item and this item is from the same group as $\tau_i$
            (b) the item on $P_j$ has utilization $\geq (1 - \frac{\ln(2)}{\gamma})/2$
            (c) $u(P_j \cup \{\tau_i\}) \leq 1 - \frac{\ln(2)}{\gamma}$
```

The following observation is crucial for our analysis and allows to link the expected waste of FFMP$^*$ to FirstFit$^*$.

**Observation 4.2.** *Consider tasks $\tau_1, \ldots, \tau_m$ such that one has $\frac{j-1}{\gamma} \leq \alpha(\tau_i) < \frac{j}{\gamma}$ (i.e. all tasks fall into the same group) and $0 \leq u(\tau_i) \leq 1 - \frac{\ln(2)}{\gamma}$ for all $i = 1, \ldots, m$. Create $m$ BinPacking items $a_1, \ldots, a_m$ with item sizes $a_i := u(\tau_i) \cdot /(1 - \ln(2)/\gamma)$, i.e. $a_i \in [0, 1]$. Then* FFMP$^*$ *schedules $\tau_1, \ldots, \tau_m$ in exactly the same way, as* FirstFit$^*$ *distributes $a_1, \ldots, a_m$, i.e. task $\tau_i$ is assigned to the $\ell$th processor if and only if item $a_i$ is assigned to the $\ell$th bin. Especially* FFMP$^*(\{\tau_1, \ldots, \tau_m\}) =$ FirstFit$^*(\{a_1, \ldots, a_m\})$.

The main result of this section will be to show that FFMP$^*(\mathcal{S}) \geq$ FFMP$(\mathcal{S})$ for any set of tasks $\mathcal{S}$. The simplicity of FFMP$^*$ will enable us to prove *monotonicity* for it, meaning that removing tasks from $\mathcal{S}$ can only lower the value of FFMP$^*(\mathcal{S})$.

**Lemma 4.3.** *For any set of tasks $\mathcal{S}$ and $\tau^* \in \mathcal{S}$ one has*

$$\texttt{FFMP}^*(\mathcal{S}) \geq \texttt{FFMP}^*(\mathcal{S} \backslash \{\tau^*\}) \geq \texttt{FFMP}^*(\mathcal{S}) - 1$$

*Proof.* Denote $\mathcal{S}' = \mathcal{S} \backslash \{\tau^*\}$ and let $\mathcal{S}_1, \ldots, \mathcal{S}_\gamma$ $[\mathcal{S}'_1, \ldots, \mathcal{S}'_\gamma]$ be the groups of $\mathcal{S}$ $[\mathcal{S}'$, resp.]. Let $i^*$ be the index such that $\tau^* \in \mathcal{S}_{i^*}$. Since the algorithm never mixes tasks from different groups one has FFMP$^*(\mathcal{S}'_i) =$ FFMP$^*(\mathcal{S}_i)$ for all $i \neq i^*$ and FFMP$^*(\mathcal{S}) = \sum_{i=1}^{\gamma}$ FFMP$^*(\mathcal{S}_i)$. Thus we may assume that all groups but $\mathcal{S}_{i^*}$ are empty. Furthermore tasks with utilization larger than $1 - \frac{\ln(2)}{\gamma}$ are never mixed with other tasks, thus their removal does not affect the assertion. Due to this we may assume that such tasks are not contained in $\mathcal{S} = \mathcal{S}_{i^*}$, hence $\mathcal{S}$ contains just tasks from the same group, all with utilization at most $1 - \frac{\ln(2)}{\gamma}$. Sticking together Observation 4.2 and the monotonicity of FirstFit$^*$ [Sho84], then yields the claim. □

By iteratively applying Lemma 4.3 we obtain

**Corollary 4.4.** FFMP$^*$ *is monotone, i.e. for all task sets $\mathcal{S}$ and $\mathcal{S}' \subseteq \mathcal{S}$ one has*

$$\texttt{FFMP}^*(\mathcal{S}) \geq \texttt{FFMP}^*(\mathcal{S}').$$

We may now conclude that the restricted variant of FFMP never produces better solutions than FFMP itself.

**Theorem 4.5.** *For all task sets $\mathcal{S}$ one has*

$$\texttt{FFMP}^*(\mathcal{S}) \geq \texttt{FFMP}(\mathcal{S}).$$

*Proof.* Let $P_1 \dot\cup \ldots \dot\cup P_m = \mathcal{S}$ be the solution, computed by FFMP and denote the groups of $\mathcal{S}$ by $\mathcal{S}_1, \ldots, \mathcal{S}_\gamma$. Consider an arbitrary processor $P_j$ and after renaming, let $\tau_1, \ldots, \tau_p$ be the tasks on $P_j$ in incoming order ($p \geq 1$). Remove $\tau_3, \ldots, \tau_p$. Given that $p \geq 2$, remove $\tau_2$ if at least one of the following conditions is true

- $\tau_1$ and $\tau_2$ stem from different groups
- $u(\tau_1) < \frac{1}{2}(1 - \frac{\ln(2)}{\gamma})$
- $u(\{\tau_1, \tau_2\}) > 1 - \frac{\ln(2)}{\gamma}$

Let $\mathcal{S}' \subseteq \mathcal{S}$ the remaining tasks. Clearly FFMP$^*$ schedules $\mathcal{S}'$ in exactly the same way that FFMP schedules them in the solution leading to FFMP$(\mathcal{S})$. Thus FFMP$^*(\mathcal{S}')$ = FFMP$(\mathcal{S})$. From Corollary 4.4 we gain FFMP$^*(\mathcal{S}) \geq$ FFMP$^*(\mathcal{S}')$. Plugging both equations/inequalities together, yields the claim. □

## 4.4 An upper bound for FFMP$^*$

In this section we will give an upper bound on the expected waste of FFMP$^*$, by exploiting the bound on the waste of FirstFit$^*$. Again Observation 4.2 will be crucial.

**Theorem 4.6.** *Let $f : \mathbb{R}_{\geq 1} \to \mathbb{R}$ be a concave and monotonic increasing function, such that $f(n)$ yields an upper bound on the expected waste of* FirstFit$^*$, *if applied to $n$ items drawn uniformly at random from* $[0, 1]$. *Then the expected waste of* FFMP$^*$ *is bounded by* $\frac{n}{\gamma} + \gamma \cdot f(n/\gamma)$ *for $n$ tasks with arbitrary periods but utilization values drawn uniformly at random from* $[0, 1]$.

*Proof.* Let $\mathcal{S}_1, \ldots, \mathcal{S}_\gamma$ be the partition of the tasks $\mathcal{S}$ into groups. Denote $n = |\mathcal{S}|$ and $n_i = |\mathcal{S}_i|$. FFMP$^*$ never mixes tasks from different groups, thus

$$\texttt{FFMP}^*(\mathcal{S}) = \sum_{i=1}^{\gamma} \texttt{FFMP}^*(\mathcal{S}_i).$$

Consider an arbitrary group $\mathcal{S}_i$. Call tasks $\tau$ with a utilization of $u(\tau) > 1 - \frac{\ln(2)}{\gamma}$ *full tasks* and *ordinary tasks* otherwise. Let $\mathcal{S}_i^{\text{full}}$ be the set of full tasks from $\mathcal{S}_i$ and let $\mathcal{S}_i' = \mathcal{S}_i \backslash \mathcal{S}_i^{\text{full}}$ be the ordinary tasks. Condition on the event $|\mathcal{S}_i'| = n_i^o$ for an arbitrary $n_i^o \in \{0, \ldots, n_i\}$. Clearly the algorithm FFMP$^*$ does not mix ordinary and full tasks, thus

$$\texttt{FFMP}^*(\mathcal{S}_i) = \texttt{FFMP}^*(\mathcal{S}_i^{\text{full}}) + \texttt{FFMP}^*(\mathcal{S}_i').$$

A full task has a utilization of at least $1 - \frac{\ln(2)}{\gamma}$, thus for each full task it suffices to account a waste of $\frac{\ln(2)}{\gamma} \leq \frac{1}{\gamma}$. The expected waste stemming from the processors, owning the full tasks of group $i$ is then

$$E[\texttt{FFMP}^*(\mathcal{S}_i^{\text{full}}) - u(\mathcal{S}_i^{\text{full}})] \leq \frac{n_i - n_i^o}{\gamma}.$$

It remains to bound the waste from the ordinary tasks. The utilization values of tasks in $\mathcal{S}_i'$ are conditioned to be in $[0, 1 - \frac{\ln(2)}{\gamma}]$. It is not difficult to see that the distribution of $u(\tau)$ for $\tau \in \mathcal{S}_i'$ is uniformly w.r.t. $[0, 1 - \frac{\ln(2)}{\gamma}]$. If we define a BinPacking instance $I_i'$ with an item of size $u(\tau)/(1 - \frac{\ln(2)}{\gamma})$ for each $\tau \in \mathcal{S}_i'$, then the item sizes in $I_i'$ are distributed uniformly w.r.t. $[0, 1]$. By Observation 4.2

$$E[\texttt{FFMP}^*(\mathcal{S}_i')] = E[\texttt{FirstFit}^*(I_i')] \leq \frac{n_i^o}{2} + f(n_i^o).$$

We can express the expected waste, stemming from the processors owning ordinary tasks from the $i$th group as

$$\begin{aligned} E[\texttt{FFMP}^*(\mathcal{S}_i') - u(\mathcal{S}_i')] \leq \left(\frac{n_i^o}{2} + f(n_i^o)\right) - E[u(\mathcal{S}_i')] &= \left(\frac{n_i^o}{2} + f(n_i^o)\right) - n_i^o \frac{1 - \ln(2)/\gamma}{2} \\ &\leq f(n_i^o) + \frac{n_i^o}{\gamma} \end{aligned}$$

The rest of the proof simply consists of summing up the achieved bounds on the waste. Combining ordinary and full tasks yields

$$E[\texttt{FFMP}^*(\mathcal{S}_i) - u(\mathcal{S}_i)] \leq \frac{n_i - n_i^o}{\gamma} + \left(f(n_i^o) + \frac{n_i^o}{\gamma}\right) \leq f(n_i) + \frac{n_i}{\gamma}$$

using monotonicity of $f$. Hence, the total expected waste for solution $\texttt{FFMP}^*(\mathcal{S})$ can be written as

$$E[\texttt{FFMP}^*(\mathcal{S}) - u(\mathcal{S})] \overset{(*)}{=} \sum_{i=1}^{\gamma} E[\texttt{FFMP}^*(\mathcal{S}_i) - u(\mathcal{S}_i)] \leq \sum_{i=1}^{\gamma} \left(f(n_i) + \frac{n_i}{\gamma}\right) \overset{(**)}{\leq} \frac{n}{\gamma} + \gamma \cdot f(n/\gamma)$$

For $(*)$ we used linearity of expectation and $(**)$ follows by Jensen's inequality and concaveness of $f$. □

Applying the best known bound on $f(n)$, we obtain

**Theorem 4.7.** *For the expected waste of* FFMP *one has*

$$E[\texttt{FFMP}(\mathcal{S}) - u(\mathcal{S})] = \mathcal{O}(n^{3/4}(\log n)^{3/8}),$$

*if $\mathcal{S}$ consists of $n$ tasks, whose utilization values are drawn uniformly at random from* $[0, 1]$.

*Proof.* Theorem 4.5 provides that bounding the waste of $\texttt{FFMP}^*$ is sufficient. Choosing $\gamma(n) := \lceil n^{1/4}/(\log n)^{3/8} \rceil$ and using the bound of $f(n) = \mathcal{O}(n^{2/3}(\log n)^{1/2})$ [Sho84] together with Theorem 4.6 yields the claim (observe that $c \cdot n^{2/3} \cdot (\log n)^{1/2}$ is concave and monotonic). □

Observing that $OPT(\mathcal{S}) = \Omega(n)$ with very high probability, we conclude that

**Corollary 4.8.** *Let $\mathcal{S}$ consist of $n$ tasks, whose utilization values are drawn uniformly at random from $[0,1]$. Then the expected approximation ratio of* `FFMP` *is*

$$E\left[\frac{\texttt{FFMP}(\mathcal{S})}{OPT(\mathcal{S})}\right] \leq 1 + \mathcal{O}(n^{-1/4}(\log n)^{3/8}).$$

## 4.5 Worst-case behaviour

In this section we outline, that the asymptotic worst-case approximation ratio is bounded by 2. Furthermore we state the worst case behaviour of `FFMP` in case that the utilization of all tasks is small. Later in Chapter 5 we will exploit this fact further. The intuition behind the next lemma is that the feasibility test $u(\mathcal{S}) \leq 1 - \beta(\mathcal{S})\ln(2)$ causes that consecutive tasks are treated like items in BinPacking. In other words, the obtained bounds correspond to that for `FirstFit`, apart from a constant additive factor. The used proof technique closely follows [BLOS95, Leu04].

**Lemma 4.9.** *Given implicit-deadline tasks $\mathcal{S} = \{\tau_1, \dots, \tau_n\}$.* `FFMP` *always produces feasible solutions and*

*(1)* $\texttt{FFMP}(\mathcal{S}) \leq 2 \cdot u(\mathcal{S}) + 4$

*(2) If $u(\tau_i) \leq \delta \leq \frac{1}{2}$ for all $i = 1, \dots, n$, then* $\texttt{FFMP}(\mathcal{S}) \leq \frac{1}{1-\delta} u(\mathcal{S}) + 3$.

*(3) Let $k \in \mathbb{N}$. If $u(\tau_i) \leq \frac{1}{2} - \frac{1}{k}$ for all $i = 1, \dots, n$, then* $\texttt{FFMP}(\mathcal{S}) \leq \frac{n}{2} + \frac{k}{2}$

*Proof.* The feasibility of the produced solution follows from the criterion of Burchard et al. (see Lemma 2.1 or [BLOS95]). Let $P_1, \dots, P_m$ be the used processors. By $\alpha(P_i) = \min\{\alpha(\tau) \mid \tau \in P_i\}$ we denote the $\alpha$-value of the first task, assigned to $P_i$ by `FFMP`.
**For (1).** Let $i \in \{1, \dots, m-1\}$ and let $\tau$ be the first task, assigned to $P_{i+1}$ for $i \in \{1, \dots, m-1\}$, thus $\alpha(\tau) = \alpha(P_{i+1})$. But the $(i+1)$th processor was only opened because $\tau$ did not fit on a prior processor. Especially it did not fit on $P_i$, thus

$$\begin{array}{rcl} u(P_i) + u(P_{i+1}) & \geq & u(P_i) + u(\tau) \\ & > & 1 - \beta(P_i \cup \{\tau\}) \cdot \ln(2) \\ & = & 1 - \ln(2) \cdot (\alpha(P_{i+1}) - \alpha(P_i)) \end{array}$$

Hence

$$\begin{array}{rcl} u(\mathcal{S}) & \geq & \sum_{i=1}^{\lfloor m/2 \rfloor} (u(P_{2i-1}) + u(P_{2i})) \\ & \geq & \sum_{i=1}^{\lfloor m/2 \rfloor} (1 - \ln(2) \cdot (\alpha(P_{2i}) - \alpha(P_{2i-1}))) \\ & \geq & \lfloor m/2 \rfloor - \ln(2) \\ & \geq & m/2 - 2 \end{array}$$

using that the differences of the $\alpha$-values sum up to at most 1. We conclude that

$$m \leq 2u(\mathcal{S}) + 4.$$

**For (2).** Again, let $\tau$ be the first task, assigned to $P_{i+1}$ for $i \in \{1, \dots, m-1\}$, then similar to (1) one has

$$u(P_i) + u(\tau) > 1 - \beta(P_i \cup \{\tau\}) \cdot \ln(2) \geq 1 - \ln(2) \cdot (\alpha(P_{i+1}) - \alpha(P_i))$$

and hence $u(P_i) \geq 1 - \delta - (\alpha(P_{i+1}) - \alpha(P_i))$. Then

$$u(\mathcal{S}) \geq \sum_{i=1}^{m-1} (1 - \delta - (\alpha(P_i) - \alpha(P_{i+1}))) \geq (m-1) \cdot (1-\delta) - 1.$$

We conclude that

$$m \leq (u(\mathcal{S}) + 1) \cdot \frac{1}{1-\delta} + 1 \leq u(\mathcal{S}) \cdot \frac{1}{1-\delta} + 3$$

**For (3).** Now consider the case that all utilizations are bounded by $\frac{1}{2} - \frac{1}{k}$. We cannot directly apply claim (1), since the utilization might tend to $n/2$, thus (1) would give us just a bound of roughly $n$. But we can use that almost each processor is owning two tasks. For $i \in \{1, \dots, m-1\}$ consider a processor $P_i$ with $|P_i| = 1$. Then it must be the case, that $2 \cdot (\frac{1}{2} - \frac{1}{k}) > 1 - (\alpha(P_{i+1}) - \alpha(P_i)) \cdot \ln(2)$ and $\alpha(P_{i+1}) - \alpha(P_i) > \frac{1}{\ln(2)} \cdot \frac{2}{k} \geq \frac{2}{k}$. Since the differences of the $\alpha$-values of the processors sum up to at most 1, this can happen at most $\lfloor k/2 \rfloor$ times. We conclude that all but $\lfloor k/2 \rfloor + 1 \leq k$ processors own two tasks, thus $n \geq 2m - k$ and $m \leq \frac{n}{2} + \frac{k}{2}$ follows. □

Note that the multiplicative factors are tight, while we did not optimize the additive ones.

## 4.6 Implementation

In the following, we explain how to implement the FFMP algorithm with a complexity of $\mathcal{O}(n \log n)$ using a heap data-structure. To this end, we rewrite the predicate of the feasibility test as follows: Consider the current task $\tau_i$ and a processor $P$. Recall that the tasks are ordered by increasing $\alpha$-values. Thus, the value of $\beta$ always depends on the current task and the task already on $P$ that defines the minimum $\alpha$-value, say $\alpha(P) = \min\{\alpha(\tau_j) \mid \tau_j \in P\}$. Hence, task $\tau_i$ can be scheduled on processor $P$ if

$$u(P) + u(\tau_i) \leq 1 - (\alpha(\tau_i) - \alpha(P)) \cdot \ln(2)$$

which is equivalent to

$$\underbrace{u(\tau_i) + \alpha(\tau_i)\ln(2)}_{v_i} \leq \underbrace{1 - u(P) + \alpha(P)\ln(2)}_{\ell_P}.$$

Note that the left-hand side (called $v_i$ in the following) only depends on the current task $\tau_i$, and that the right-hand side (called $\ell_P$ in the following) only depends on the tasks which are already scheduled on the processor $P$. For $P = \emptyset$, we define $\ell_P = \infty$. Hence, we may maintain a binary heap data-structure to find in logarithmic time the processor with the least index that passes the feasibility test. Viewed as a binary tree, we have $n$ leaves corresponding to processors. They are labeled with the right-hand sides $\ell_P$ depending on their current utilization and the $\alpha$-value of the first task which was scheduled on each one. Initially, they are empty and their labels are $\infty$. The other nodes of the tree are labeled with the maximum label of their respective children. See Figure 4.6 for a visualisation. Starting from the root, we proceed with the left child, if the $v_i$ value for the current task is not greater than the label of the left child. Otherwise, we turn to the right child, which then has a sufficiently large label by construction. The leaf that we eventually reach, determines the processor on which we schedule the current task. Since the height of the binary tree is logarithmic in the number of leaves, i.e. $n$, we can schedule each task in $\mathcal{O}(\log n)$. Moreover, the update of the data-structure also takes $\mathcal{O}(\log n)$, since we only need to traverse the tree back to the root and update the labels on this path; any other label remains invariant.

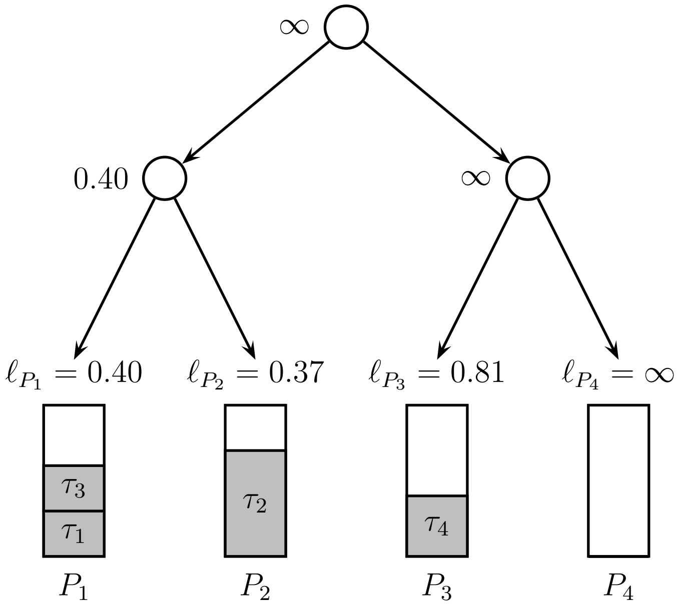


Figure 4.6: Example for the heap structure. We use notation $\tau_i = (u(\tau_i), \alpha(\tau_i))$. Then `FFMP` schedules tasks $\tau_1 = (0.3, 0.0), \tau_2 = (0.7, 0.1), \tau_3 = (0.3, 0.2), \tau_4 = (0.4, 0.3)$ as depicted.

# Chapter 5

# Beating the Worst-case Approximation Ratio

We again deal with Rate-monotonic multiprocessor scheduling of a set $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$ of implicit-deadline tasks, i.e. each task $\tau_i = (c(\tau_i), p(\tau_i))$ releases a job of length $c(\tau_i)$ at each integer multiple of the period $p(\tau_i)$. In this chapter we beat the asymptotic 7/4-approximation algorithm of the *Rate-monotonic General Task* algorithm (`RMGT`) of Burchard et al. [BLOS95] by obtaining an asymptotic 3/2-approximation based on matching techniques.

Let us start with describing the `RMGT` algorithm (see Algorithm 5). It splits the instance $\mathcal{S}$ into *small tasks* with utilization at most 1/3 and the remaining *large tasks* of utilization larger than 1/3. The small tasks are distributed in a Next Fit manner using the sufficient feasibility criterion $u(\mathcal{S}') \leq 1 - \beta(\mathcal{S}')\ln(2)$ for $\mathcal{S}' \subseteq \mathcal{S}$. The large tasks are distributed in a First Fit manner, where one exploits that there is a simple exact RM-schedulability test for 2 tasks (see Section 2.2). Consider the obtained solution. We call a processor

- *small* if it owns small tasks
- *1-processor* if it owns a single large task
- *2-processor* if it owns 2 large tasks.

Then one can show that the average utilization of the small processors, 1-processors and 2-processors must be at least $2/3 - o(1)$, $1/2 - o(1)$ and $2/3$, respectively. Since we use the

**Algorithm 5** Rate-monotonic general task algorithm (`RMGT`) [BLOS95]

**Input:** Set $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$ of implicit-deadline tasks
**Output:** Assignment of $\tau_1, \ldots, \tau_n$ to processors $P_1, P_2, \ldots$

```
1. Split $\mathcal{S}$ into $\mathcal{S}_s = \{\tau \in \mathcal{S} \mid u(\tau) \leq 1/3\}$ and $\mathcal{S}_\ell = \{\tau \in \mathcal{S} \mid u(\tau) > 1/3\}$.
2. Sort $\mathcal{S}_s$ by increasing $\alpha$-values.
3. Distribute tasks in $\mathcal{S}_s$ via Next Fit where $\mathcal{S}' \subseteq \mathcal{S}_s$ is considered as feasible, if
   $u(\mathcal{S}') \leq 1 - \beta(\mathcal{S}')\ln(2)$.
4. Distribute tasks in $\mathcal{S}_\ell$ via First Fit, using the exact feasibility test for 2 tasks.
5. Output the union of both schedules.
```

exact feasibility test for distributing the large tasks, the number of 1-processors yields a lower bound on $OPT$. Plugging together these obtained bounds then yields the claimed asymptotic approximation guarantee of 7/4.

We now wonder, how this algorithm could be improved. Observe that for the sub-instance of large tasks we can even compute an optimum solution using *matchings*. It lies on hand to use this fact for an approximation algorithm that improves over the ratio of 7/4.

## 5.1 A matching based algorithm

Let $G = (V, E)$ be an undirected graph. A *matching* is a subset of edges, which are pairwise disjoint. The MinCostMatching problem is defined as

MinCostMatching
Given: An undirected graph $G = (V, E)$ with edge costs $c : E \to \mathbb{Q}$.
Find:

$$\min \left\{ \sum_{e \in M} c(e) \mid M \text{ is a matching} \right\}$$

Note that edge costs might be negative, since otherwise the empty matching would be optimal. The MinCostMatching problem can be solved in polynomial time using Edmonds' famous *Blossom* algorithm [Edm65b, Edm65a]. This algorithm can be implemented to run in time $\mathcal{O}(n^3)$ [Sch03] and even in time $\mathcal{O}(n(m + n \log n))$ for graphs with $m$ edges, see Gabow [Gab90]. We refer to the books of Cook, Cunningham, Pulleyblank & Schrijver [CCPS97] and of Schrijver [Sch03] for an extensive account on matchings.

We define an undirected graph $G = (\mathcal{S}, E)$, whose nodes are the tasks and which contains an edge $\{\tau_1, \tau_2\} \in E$, if and only if the set $\{\tau_1, \tau_2\}$ of tasks is RM-schedulable on a single processor. Furthermore we have edge weights $c : E \to \mathbb{Q}$ and node weights $w : \mathcal{S} \to \mathbb{Q}$, for which we will give a suitable choice in a few sentences. It is our aim that a minimum cost matching in this graph yields a good multiprocessor schedule. Here it will pay off, to define the cost of a matching $M \subseteq E$ in a non-standard way as the cost of the edges in $M$ plus the cost of the nodes, which are *not* covered by $M$. Formally

$$c(M) := \sum_{e \in M} c(e) + \sum_{\tau \in \mathcal{S} \setminus M} w(\tau)$$

where, by a slight abuse of notation $\mathcal{S} \setminus M := \{\tau \in \mathcal{S} \mid \tau \text{ not covered by } M\}$.

We now want to determine edge and node weights, such that any matching $M$ of cost, say $\delta$, can be turned into a feasible multiprocessor schedule using essentially $\delta$ many processors.

By definition, two tasks $\{\tau_1, \tau_2\} = e \in M$ can be scheduled on a single processor, thus we choose unit edge costs, i.e. $c(e) := 1$ for all $e \in E$. We want to distribute tasks that not covered by $M$ with `FFMP`. Recall that the `FFMP` algorithm distributes tasks in a simple First Fit manner, respecting the following bounds (see Lemma 4.9 in the previous chapter):

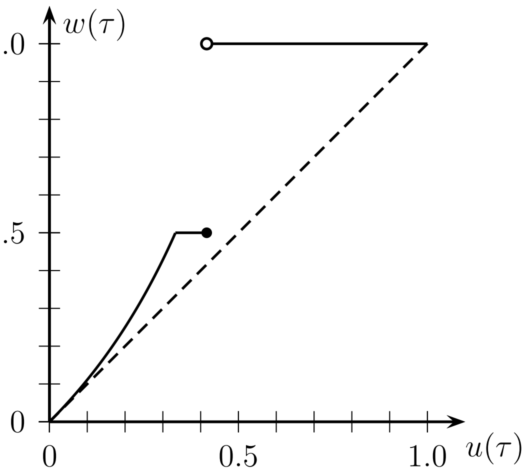


Figure 5.1: Graph of node weights $w(\tau)$ for $k = 1$.

- $\texttt{FFMP}(\mathcal{S}) \leq \frac{1}{1-\max_{\tau\in\mathcal{S}} u(\tau)} u(\mathcal{S}) + 3$.
- Let $q \in \mathbb{N}$ and $u(\tau) \leq \frac{1}{2} - \frac{1}{q}$ for all $i = 1, \ldots, n$. Then $\texttt{FFMP}(\mathcal{S}) \leq \frac{n}{2} + \frac{q}{2}$.

We define the node weights depending on the utilization. Here we distinguish 3 categories of tasks:

- *Small tasks* ($0 \leq u(\tau) \leq \frac{1}{3}$): Consider tasks $\tau_1, \ldots, \tau_m$ with a small utilization, i.e. $u(\tau_i) \leq u_{\max}$ for all $i = 1, \ldots, m$ and $u_{\max} \leq 1/3$. Then we may schedule such tasks with FFMP using

  $$u(\{\tau_1, \ldots, \tau_m\}) \frac{1}{1 - u_{\max}} + 3 \leq m \cdot u_{\max} \frac{1}{1 - u_{\max}} + 3$$

  many processors, thus we choose $w(\tau) := \frac{u(\tau)}{1-u(\tau)}$ for a small task $\tau$.
- *Medium tasks* ($\frac{1}{3} < u(\tau) \leq \frac{1}{2} - \frac{1}{12k}$): Suppose we have tasks $\tau_1, \ldots, \tau_m$ whose utilization is at least 1/3, but bounded away from 1/2, say $u(\tau_i) \leq \frac{1}{2} - \frac{1}{12k}$, where $k$ is an integer parameter that we determine later. Then $\texttt{FFMP}(\{\tau_1, \ldots, \tau_m\}) \leq m/2 + \mathcal{O}(k)$, thus we choose $w(\tau) := 1/2$ for medium tasks.
- *Large tasks* ($u(\tau) > \frac{1}{2} - \frac{1}{12k}$): For a large task one processor is sufficient and possibly needed, thus $w(\tau) := 1$ in this case.

For a choice of $k = 1$ the graph of $w$ can be found in Figure 5.1.

We now have all ingredients that we need for our algorithm RMMatching: We first compute an optimum matching $M$ w.r.t. weights $c$ and $w$. Each edge in $M$ yields a pair of tasks schedulable together on a single processor. The uncovered tasks are distributed by FFMP. See Algorithm 6 for a formal description.

In our first proof we derive a polynomial bound on the running time.

**Lemma 5.1.** *The* RMMatching *algorithm can be implemented to run in time $\mathcal{O}(n^3)$.*

**Algorithm 6** `RMMatching`

**Input:** Set $\mathcal{S} = \{\tau_1, \dots, \tau_n\}$ of implicit-deadline tasks; parameter $k \in \mathbb{N}$

**Output:** Assignment of $\tau_1, \dots, \tau_n$ to processors $P_1, P_2, \dots$

1. Construct $G = (\mathcal{S}, E)$ with edges $(\tau_1, \tau_2) \in E \Leftrightarrow \{\tau_1, \tau_2\}$ RM-schedulable. Choose edge costs $c(e) = 1$ and vertex costs

$$w(\tau) = \begin{cases} u(\tau) \cdot \frac{1}{1-u(\tau)} & \text{if } u(\tau) \leq \frac{1}{3} \\ \frac{1}{2} & \text{if } \frac{1}{3} < u(\tau) \leq \frac{1}{2} - \frac{1}{12k} \\ 1 & \text{if } u(\tau) > \frac{1}{2} - \frac{1}{12k} \end{cases}$$

2. Find a matching $M \subseteq E$ minimizing

$$\sum_{e \in M} c(e) + \sum_{\tau \in \mathcal{S} \setminus M} w(\tau)$$

3. For all $\{\tau_1, \tau_2\} \in M$ create a processor with tasks $\{\tau_1, \tau_2\}$
4. Define
   - $\mathcal{S}_i = \{\tau \in \mathcal{S} \setminus M \mid \frac{1}{3} \cdot \frac{i-1}{k} \leq u(\tau) < \frac{1}{3} \cdot \frac{i}{k}\} \ \forall i = 1, \dots, k$
   - $\mathcal{S}_{k+1} = \{\tau \in \mathcal{S} \setminus M \mid \frac{1}{3} \leq u(\tau) \leq \frac{1}{2} - \frac{1}{12k}\}$
   - $\mathcal{S}_{k+2} = \{\tau \in \mathcal{S} \setminus M \mid u(\tau) > \frac{1}{2} - \frac{1}{12k}\}$
5. Distribute $\mathcal{S}_{k+2}, \mathcal{S}_{k+1}, \dots, \mathcal{S}_1$ via `FFMP`.

*Proof.* For each set $\{\tau_1, \tau_2\}$, RM-schedulability can be tested in constant time [Leu04], thus the graph $G$ can be constructed in $\mathcal{O}(n^2)$. The running time of `FFMP` is $O(n' \log n')$ for scheduling $n'$ tasks, thus the total running time spent for Step (5) of the algorithm is certainly upper bounded by $\mathcal{O}(n^3)$.

It remains to show, how a min-cost matching $M$ can be computed. But the cost of a matching $M$ can be rewritten as

$$\sum_{e \in M} c(e) + \sum_{\tau \in \mathcal{S} \setminus M} w(\tau) = \sum_{\{\tau_1,\tau_2\} \in M} (c(\tau_1, \tau_2) - w(\tau_1) - w(\tau_2)) + \sum_{\tau \in \mathcal{S}} w(\tau).$$

Since the second summand $\sum_{\tau \in \mathcal{S}} w(\tau)$ does not depend on $M$, this expression can be minimized by a single MinCostMatching application with edge costs $c'(\tau_1, \tau_2) := c(\tau_1, \tau_2) - w(\tau_1) - w(\tau_2)$. The claim then follows. □

In a next step, we derive an upper bound on the approximation ratio.

**Theorem 5.2.** *The* `RMMatching` *algorithm produces a solution of cost at most* $(\frac{3}{2} + \frac{1}{k}) OPT_{\text{MulSched}} + 9k$.

*Proof.* Let $P_1, \ldots, P_m$ with $m = OPT_{\text{MulSched}}(\mathcal{S})$ be an optimum solution for task set $\mathcal{S}$. Now we will prove that

**(I)** There is a matching solution of cost at most $(\frac{3}{2} + \frac{1}{12k}) OPT_{\text{MulSched}}$.

**(II)** `RMMatching` turns a matching $M$ of cost $\delta$ into an MulSched solution of at most $(1 + \frac{1}{2k})\delta + 9k$ many processors.

The claim then follows from (I)+(II), since

$$\left(1 + \frac{1}{2k}\right) \cdot \left(\frac{3}{2} + \frac{1}{12k}\right) OPT_{\text{MulSched}} + 9k \leq \left(\frac{3}{2} + \frac{1}{k}\right) OPT_{\text{MulSched}} + 9k.$$

**Part (I).** Consider a processor $P_i$. After reordering let $\tau_1, \ldots, \tau_q$ be the tasks on $P_i$, sorted such that $u(\tau_1) \geq \ldots \geq u(\tau_q)$. First suppose that $q \geq 2$. We will either cover the two tasks with highest utilization in $P_i$ by a matching edge or leave all tasks from $P_i$ uncovered. But in any case we guarantee, that the tasks in $P_i$ contribute at most $\frac{3}{2} + \frac{1}{12k}$ to the cost of the matching.

- Case: $u(\tau_1) \leq 1/3$. We leave all tasks in $P_i$ uncovered. This contributes at most

$$\sum_{j=1}^{q} w(\tau_j) = \underbrace{\sum_{j=1}^{q} u(\tau_j)}_{\leq 1} \cdot \overbrace{\frac{1}{1 - \underbrace{u(\tau_j)}_{\leq 1/3}}}^{\leq 3/2} \leq \frac{3}{2}$$

- Case: $u(\tau_2) \leq \frac{1}{3} < u(\tau_1) \leq \frac{1}{2} - \frac{1}{12k}$. Again we do not cover any task by a matching edge. The contribution is

$$\underbrace{w(\tau_1)}_{=1/2} + \sum_{j=2}^{q} w(\tau_j) = \frac{1}{2} + \underbrace{\sum_{j=2}^{q} u(\tau_j)}_{\leq 2/3} \cdot \overbrace{\frac{1}{1 - \underbrace{u(\tau_j)}_{\leq 1/3}}}^{\leq 3/2} \leq \frac{3}{2}$$

- Case: $u(\tau_1) > \frac{1}{2} - \frac{1}{12k}$. We add $\{\tau_1, \tau_2\}$ to the matching and leave $\tau_3, \ldots, \tau_q$ uncovered. The contribution is

$$1 + \underbrace{\sum_{j=3}^{q} u(\tau_j)}_{\leq 1 - u(\{\tau_1,\tau_2\})} \cdot \frac{1}{1 - \underbrace{u(\tau_j)}_{\leq u(\tau_2)}} \leq 1 + \frac{1 - u(\tau_1) - u(\tau_2)}{1 - u(\tau_2)} \leq 2 - u(\tau_1) \leq \frac{3}{2} + \frac{1}{12k}.$$

- Case: $\frac{1}{3} < u(\tau_2) \leq u(\tau_1) \leq \frac{1}{2} - \frac{1}{12k}$. We again add $\{\tau_1, \tau_2\}$ to the matching. The contribution is

$$1 + \sum_{j=3}^{q} w(\tau_j) = 1 + \underbrace{\sum_{j=3}^{q} u(\tau_j)}_{\leq 1/3} \cdot \overbrace{\frac{1}{1 - \underbrace{u(\tau_j)}_{\leq 1/3}}}^{\leq 3/2} \leq \frac{3}{2}$$

  since $u(\{\tau_1, \tau_2\}) \geq 2/3$.

If $q = 1$, then we do not cover $\tau_1$. The contribution is at most 1.

**Part (II).** It remains to show, that a matching solution $M$ of cost $\delta$ is turned into a MulSched solution, consisting of at most $(1 + \frac{1}{2k})\delta + 9k$ processors. Recall that

$$\delta = |M| + \sum_{\tau \in \mathcal{S}_1 \cup \ldots \cup \mathcal{S}_{k+2}} w(\tau).$$

We create $|M|$ processors, covering pairs of tasks $\{\tau_1, \tau_2\} \in M$. For scheduling the tasks in $\mathcal{S}_{k+1}$ we know that according to Lemma 4.9

$$\texttt{FFMP}(\mathcal{S}_{k+1}) \leq \frac{|\mathcal{S}_{k+1}|}{2} + \frac{12k}{2} = \sum_{\tau \in \mathcal{S}_{k+1}} w(\tau) + 6k,$$

using that the utilization of all tasks in $\mathcal{S}_{k+1}$ lies between $\frac{1}{3}$ and $\frac{1}{2} - \frac{1}{12k}$. Of course $\texttt{FFMP}(\mathcal{S}_{k+2}) \leq |\mathcal{S}_{k+2}| = \sum_{\tau \in \mathcal{S}_{k+2}} w(\tau)$. Let $i \in \{1, \ldots, k\}$. For $\mathcal{S}_i$ we know that the utilization of each task in $\mathcal{S}_i$ is sandwiched between $\frac{1}{3} \cdot \frac{i-1}{k}$ and $\frac{1}{3} \cdot \frac{i}{k}$. Consequently

$$\texttt{FFMP}(\mathcal{S}_i) \leq \frac{1}{1 - \frac{1}{3} \cdot \frac{i}{k}} u(\mathcal{S}_i) + 3 \leq \frac{1 + \frac{1}{2k}}{1 - \frac{1}{3} \cdot \frac{i-1}{k}} \cdot u(\mathcal{S}_i) + 3 \leq \left(1 + \frac{1}{2k}\right) \sum_{\tau \in \mathcal{S}_i} w(\tau) + 3$$

using again Lemma 4.9. We conclude, that the total number of processors in the produced solution is bounded by

$$\begin{aligned} |M| + \sum_{i=1}^{k+2} \texttt{FFMP}(\mathcal{S}_i) &\leq |M| + \left(1 + \frac{1}{2k}\right) \sum_{\mathcal{S}_1 \cup \ldots \cup \mathcal{S}_{k+2}} w(\tau) + 3k + 6k \\ &\leq \left(1 + \frac{1}{2k}\right) \delta + 9k. \end{aligned}$$

The claim then follows. □

We conclude

**Corollary 5.3.** *For any choice of $k = \omega(1)$ and $k = o(u(\mathcal{S}))$,* `RMMatching` *is an asymptotic $3/2$-approximation algorithm.*

We furthermore see, that the algorithm produces a near optimal solution, if all tasks are large.

**Corollary 5.4.** *If $u(\tau) > 1/3$ for all tasks $\tau \in \mathcal{S}$, then* `RMMatching` *computes a solution of cost at most*

$$OPT_{\text{MulSched}}(\mathcal{S}) + 6k.$$

*Proof.* Let $P_1, \ldots, P_m$ be an optimum multiprocessor schedule for $\mathcal{S}$. Then there is a matching $M$ of cost at most $m$, by taking the paired tasks as matching edges. Note that $\mathcal{S}_1 = \ldots = \mathcal{S}_k = \emptyset$, since $u(\tau) > 1/3$ for all tasks. Similar to the proof of Theorem 5.2 we see, that the produced MulSched solution will have cost of at most

$$|M| + \texttt{FFMP}(\mathcal{S}_{k+1}) + \texttt{FFMP}(\mathcal{S}_{k+2}) \leq |M| + \sum_{\tau \in \mathcal{S}_{k+1} \cup \mathcal{S}_{k+2}} w(\tau) + 6k \leq m + 6k.$$

This concludes the assertion. □

*Remark* 5.5. For large instances, the running time of $\mathcal{O}(n^3)$ might not be acceptable for application purposes. We now outline, how to speed up the algorithm without significantly worsen the quality of the solutions in practice. On the other hand, we will not be able anymore to guarantee the asymptotic worst-case approximation ratio of 3/2.

Consider the graph $G$, constructed in the algorithm and redefine the node weights $w(\tau)$, such that $w(\tau) = 1/2$ for $1/3 \leq u(\tau) \leq 1/2$. We may now delete all edges $e$ with $c'(\tau_1, \tau_2) = c(\tau_1, \tau_2) - w(\tau_1) - w(\tau_2) \geq 0$ from the graph $G$, since they are not needed for a min-cost matching. For each remaining edge $e = (\tau_1, \tau_2)$ either $\tau_1$ or $\tau_2$ must be large (i.e. $u(\tau_1) > 1/2$ or $u(\tau_2) > 1/2$). Consequently $G$ is bipartite and a min-cost matching can be found very efficiently either by the so called *Hungarian method* (see e.g. [Sch03]) or by using a powerful LP solver like CPLEX, applied to the standard matching LP formulation [Sch03]. To further increase the performance one can make the graph sparse, by removing all but the $m$ weight-minimal edges (w.r.t. $c'$) adjacent to $\tau$, for each $\tau \in \mathcal{S}$ with $u(\tau) > 1/2$. Here $m$ can be chosen as a small constant, say $m = 10$.

## 5.2 Average-case behaviour

In many cases the provable guarantee of approximation algorithms is at the expense of the practical performance. However, we believe that this is not the case for `RMMatching`. Using the results from Chapter 4 we can show with very little effort that our algorithm has an expected waste of $\mathcal{O}(n^{3/4}(\log n)^{3/8})$ (the same bound as for `FFMP`) if the input consists of $n$ tasks with arbitrary periods, but utilization values drawn uniformly at random from $[0, 1]$.

Recall that the *waste* of a solution is defined as the number of processors minus the utilization. In Chapter 4 we proved that the $\mathcal{O}(n^{3/4}(\log n)^{3/8})$ upper bound on the expected waste even holds for the restricted algorithm `FFMP*`, which never assigns more than two tasks per processor. It immediately follows that

**Lemma 5.6.** *Let $OPT^2_{\text{MulSched}}(\mathcal{S})$ be the minimum number of processors, needed to RM-schedule $\mathcal{S}$, such that never more than 2 tasks are assigned to the same processor. If $\mathcal{S} = \{\tau_1, \dots, \tau_n\}$ and each $u(\tau_i)$ is drawn uniformly at random from $[0,1]$, then*

$$E\left[OPT^2_{\text{MulSched}}(\mathcal{S})\right] \leq E[u(\mathcal{S})] + \mathcal{O}(n^{3/4}(\log n)^{3/8}).$$

With this result at hand, the following proof is easy to obtain.

**Lemma 5.7.** *Assign arbitrary periods to $n$ tasks $\mathcal{S} = \{\tau_1, \dots, \tau_n\}$ and draw $u(\tau_i) \in [0,1]$ uniformly at random. For a parameter $k = \lceil \sqrt{n} \rceil$ one has*

$$E[\texttt{RMMatching}(\mathcal{S})] \leq E[u(\mathcal{S})] + \mathcal{O}(n^{3/4}(\log n)^{3/8}).$$

*Proof.* Considering the definition of $G$ in `RMMatching` we observe that there is a matching solution of cost at most $OPT^2_{\text{MulSched}}(\mathcal{S})$. Recalling part (II) of the proof of Theorem 5.2, we know that the algorithm produces a solution of expected cost at most

$$\begin{aligned} & E\left[\left(1 + \frac{1}{2k}\right) OPT^2_{\text{MulSched}}(\mathcal{S}) + 9k\right] \\ \leq\ & \left(1 + \frac{1}{2\sqrt{n}}\right) \cdot \left(E[u(\mathcal{S})] + \mathcal{O}(n^{3/4}(\log n)^{3/8})\right) + 9\lceil \sqrt{n} \rceil \\ \leq\ & E[u(\mathcal{S})] + \mathcal{O}(n^{3/4}(\log n)^{3/8}) \end{aligned}$$

using that $E[u(\mathcal{S})] = n/2$. □

# Chapter 6

# A Column-based Linear Programming Relaxation

As a standard technique in discrete optimization we will consider the *integer linear program* for the MulSched problem and the corresponding *linear programing (LP) relaxation.* Again let $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$ be implicit-deadline tasks and we aim at finding a partition of $\mathcal{S}$ into RM-schedulable systems $\mathcal{S}_1, \ldots, \mathcal{S}_k$ with $k$ minimal.

In this chapter we obtain that the asymptotic integrality gap of the column-based MulSched LP relaxation is sandwiched between $4/3 \approx 1.33$ and $1 + \ln(2) \approx 1.69$. To the best of our knowledge, this is the first result dealing with this LP relaxation.

## 6.1 The LP relaxation for Bin Packing

We remember that MulSched is a generalization of BinPacking, thus it will again pay off, to first consider the well-studied LP relaxations for the latter problem. Suppose we are given a BinPacking instance $\mathcal{I} = \{a_1, \ldots, a_n\}$ with item sizes $0 \leq a_i \leq 1$. The intuitive way to model such a problem as an integer linear program is by introducing binary decision variables $x_{ij}$, which are 1 if and only if item $a_i$ is assigned to bin $j$. But this formulation allows that items may be arbitrarily split, consequently the optimum fractional value always equals the trivial lower bound of $\text{size}(\mathcal{I}) = \sum_{i:a_i \in \mathcal{I}} a_i$. Such a formulation is therefore not helpful.

This is the motivation for introducing the *Gilmore-Gomory LP formulation* [GG61], which is *column-based.* We call $x \in \{0,1\}^n$ a *pattern* if $x$ denotes a feasible way to pack a single bin, i.e. if $a^T x \leq 1$. Let $\mathcal{P}(\mathcal{I}) = \{x \in \{0,1\}^n \mid a^T x \leq 1\}$ be the set of all feasible patterns. The LP relaxation is then

$$\begin{aligned} \min \mathbf{1}^T \lambda & \qquad (BLP) \\ \sum_{x \in \mathcal{P}} \lambda_x x &\geq \mathbf{1} \\ \lambda &\geq \mathbf{0} \end{aligned}$$

where $\lambda_x$ is a variable, telling us, whether to use a pattern $x$. Let $OPT^f_{\text{BinPacking}}(\mathcal{I})$ be the optimum value of this LP. In general this linear program has an exponential number of variables, but only $n$ constraints. That means one might have a chance to solve the dual, which has only a polynomial number of variables. The dual is

$$\begin{array}{rcl} \max \mathbf{1}^T w & & (DLP) \\ w^T x & \leq & 1 \quad \forall x \in \mathcal{P} \\ w & \geq & \mathbf{0} \end{array}$$

The *Ellipsoid algorithm* [Kha79, GLS81] can find an optimum solution to $(DLP)$ in polynomial time, if we can separate the set of solutions, i.e. if for a given vector $w$, we can decide in polynomial time whether $w$ is feasible or if not, yield a violated constraint. Hence, we have to solve the following separation problem

$$\begin{array}{rcl} \max \sum_{i=1}^{n} x_i w_i & & \\ \sum_{i=1}^{n} a_i x_i & \leq & 1 \\ x & \in & \{0,1\}^n \end{array}$$

which is exactly the Knapsack problem with item sizes $a_i$ and profits $w_i$.

Knapsack
Given: Profits $c_1, \ldots, c_n \in \mathbb{Q}_+$, weights $a_1, \ldots, a_n \in \mathbb{Q}_+$, bound $B > 0$
Find:

$$\max \left\{ \sum_{i \in I} c_i \mid I \subseteq \{1, \ldots, n\} : \sum_{i \in I} a_i \leq B \right\}$$

Although Knapsack is weakly **NP**-hard [GJ79, Weg05], there is an efficient FPTAS available [KPP04, Vaz01]. That means we can solve the separation oracle approximately. Consequently for any $\varepsilon > 0$, we can find a solution for the Gilmore-Gomory LP relaxation of cost at most $(1+\varepsilon)OPT^f_{\text{BinPacking}}(\mathcal{I})$ in time polynomial in $n$ and in $1/\varepsilon$ [GLS81]. For example we can even find a fractional basic solution of cost at most $OPT^f_{\text{BinPacking}}(\mathcal{I})+1$.

The *asymptotic integrality gap* is defined as

$$\limsup_{OPT(\mathcal{I}) \to \infty} \frac{OPT(\mathcal{I})}{OPT_f(\mathcal{I})}$$

An outstanding result of Karmarkar & Karp is that this gap is 1 for BinPacking. More precisely

**Theorem 6.1.** *[KK82] For any* BinPacking *instance consisting of $n$ items, one has*

$$OPT_{\text{BinPacking}} - \lceil OPT^f_{\text{BinPacking}} \rceil = \mathcal{O}(\log^2 n).$$

Moreover the result is constructive, i.e. an integral solution of cost $OPT^f_{\text{BinPacking}} + \mathcal{O}(\log^2 n)$ can be found in polynomial time. This still holds if $n$ is the number of *different* item sizes and the LP is equipped with a general right hand side

$$\begin{array}{rcl} \min \mathbf{1}^T \lambda & & \\ \sum_{x \in \mathcal{P}} \lambda_x x & \geq & b \\ \lambda & \geq & \mathbf{0} \end{array}$$

Here $b_i \in \mathbb{N}$ denotes the multiplicity of item type $a_i$. One can briefly sketch the Karmarkar-Karp algorithm as follows

1. Compute a near optimal basic solution $\lambda$ to $(BLP)$
2. Use $\lfloor \lambda_x \rfloor$ times pattern $x$
3. Create a dominating instance $\mathcal{I}'$ with $n/2$ many different item sizes and recurse.

A basic feasible solution to $(BLP)$ has at most $n$ patterns $x$ with $\lambda_x > 0$. Consequently the cumulated size of the items, which remain after step (2) is at most $n$. Then by rounding item sizes, the number of different item types is decreased to $n/2$, while this increases the value of $OPT^f_{\text{BinPacking}}$ by not more than $\mathcal{O}(\log n)$. The promised bound follows by observing that the recursion depth is $\mathcal{O}(\log n)$.

An even stronger claim is conjectured:

**Conjecture.** *(Mixed Integer Roundup Conjecture). For all* BinPacking *instances one has* $OPT_{\text{BinPacking}} - \lceil OPT^f_{\text{BinPacking}} \rceil \leq 1$.

Clearly we wonder, whether similar properties also hold for the more general case of MulSched. We will now see, that this is not true.

## 6.2 The LP relaxation for multiprocessor scheduling

Let $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$ be a set of implicit-deadline tasks. In the same manner as for BinPacking, a *pattern* $x = \chi(\mathcal{S}')$ is now redefined as the characteristic vector of an RM-schedulable set $\mathcal{S}' \subseteq \mathcal{S}$. Furthermore denote

$$\mathcal{P} := \mathcal{P}(\mathcal{S}) = \{x \in \{0,1\}^n \mid \{\tau_i \in \mathcal{S} \mid x_i = 1\} \text{ is RM-schedulable}\}$$

again as the set of all patterns. Consider the linear program $(MLP)$

$$\min \left\{ \mathbf{1}^T \lambda \mid \sum_{x \in \mathcal{P}} \lambda_x x \geq \mathbf{1},\ \lambda \geq \mathbf{0} \right\}$$

whose optimum value is termed $OPT^f_{\text{MulSched}}(\mathcal{S})$. To obtain at least a $(1+\varepsilon)$-approximate solution to $(MLP)$ we need to solve the following separation problem:

MulSched-Separation
Given: Set $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$ $(\tau_i = (c(\tau_i), p(\tau_i)))$ of implicit-deadline tasks; profits $w(\tau_1), \ldots, w(\tau_n) \in \mathbb{Q}_+$
Find:

$$\max \left\{ \sum_{\tau \in \mathcal{S}'} w(\tau) \mid \mathcal{S}' \subseteq \mathcal{S} \text{ is RM-schedulable} \right\}$$

As a generalisation of Knapsack, this problem is at least weakly **NP**-hard. Unfortunately we do not know, whether one can also approximate this problem as good as Knapsack.

**Open Problem 6.2.** *Is there an FPTAS for* MulSched-Separation*?*

Fortunately we obtained in Chapter 3, that tasks can be merged, s.t. for approximating MulSched it suffices to obtain an algorithm that assumes a lower bound on the utilization of each task. If $u(\tau) \geq \varepsilon$ for all $\tau \in \mathcal{S}$, then each pattern contains at most $\lfloor 1/\varepsilon \rfloor$ tasks, thus the number of possible patterns can be bounded by $|\mathcal{P}| \leq \binom{n}{\lfloor 1/\varepsilon \rfloor} \leq n^{1/\varepsilon}$, which is a polynomial for fixed $\varepsilon > 0$. Then $(MLP)$ has a polynomial number of variables and constraints, thus an optimum solution can be determined using any polynomial time linear programming solver like the Ellipsoid algorithm of Khachiyan [Kha79] or the Interior Point method of Karmarkar [Kar84].

We now determine bounds on the value of the integrality gap.

## A lower bound

Let us begin with obtaining the result, that the LP relaxation for MulSched is much weaker than that of BinPacking.

**Theorem 6.3.** *$(MLP)$ has an asymptotic integrality gap of at least $4/3$.*

*Proof.* We state an instance $\mathcal{S}$ with $3n$ tasks, having

$$\frac{OPT_{\text{MulSched}}}{OPT^{f}_{\text{MulSched}}} \geq \frac{2n}{3/2 \cdot n} = \frac{4}{3}.$$

Choose periods $1 < p_1 < p_2 < \ldots < p_n < 2$ arbitrarily. For notational convenience we may consider $\mathcal{S}$ as a multi-set, containing $b_i = 3$ times task $\tau_i$ with $(c(\tau_i), p(\tau_i)) = (p_i/2, p_i)$ for all $i = 1, \ldots, n$. Note that $u(\tau_i) = 1/2$ for each task $\tau_i$. Let $x_i$ be the pattern, containing task $\tau_i$ exactly twice. Taking each pattern $x_i$ exactly $\lambda_{x_i} = \frac{3}{2}$ times yields a feasible fractional solution of cost $\frac{3}{2}n$.

We next demonstrate, that never two copies of different tasks can be scheduled together on the same processor. Let $\tau, \tau'$ denote tasks with $p(\tau') < p(\tau)$ (thus $c(\tau)/2 < c(\tau') < c(\tau)$)

Task $\tau$ is feasible if and only if there is an $r > 0$, s.t.

$$c(\tau) + \left\lceil \frac{r}{p(\tau')} \right\rceil c(\tau') \leq r \leq p(\tau)$$

We will lead this to a contradiction.
Case $r > p(\tau')$: Then

$$c(\tau) + \underbrace{\left\lceil \frac{r}{p(\tau')} \right\rceil}_{\geq 2} \underbrace{c(\tau')}_{> \frac{1}{2} c(\tau)} > 2c(\tau) = p(\tau)$$

leading to a contradiction since $\frac{1}{2} < c(\tau_1) < \ldots < c(\tau_n) < 1$.
Case $r \leq p(\tau')$: Then one has

$$\underbrace{c(\tau)}_{> c(\tau')} + \underbrace{\left\lceil \frac{r}{p(\tau')} \right\rceil}_{=1} c(\tau') > 2c(\tau') = p(\tau') \geq r.$$

Again we have a contradiction. We obtain, that for each 3 copies of $\tau_i$, we need 2 processors, thus $OPT_{\textsc{MulSched}} \geq 2n$. The claim then follows. □

## An upper bound

In a next step, we derive an upper bound on the integrality gap, using randomized rounding. Let $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$ be an arbitrary instance, $\lambda$ be a solution for $(MLP)$ and let $\mathcal{P}(\mathcal{S}) = \{x^1, \ldots, x^K\}$ be the patterns, belonging to the instance. That means, the fractional solution contains pattern $x^i$ to the extent of $\lambda_i \geq 0$. Let $\alpha \in [0, 1]$ be a parameter, that we are going to determine later. Independently from each other, we *buy* pattern $x^i$ with probability $\alpha \cdot \lambda_i$, where buying means that we fill a processor with the tasks contained in $x^i$. Consider an arbitrary task $\tau \in \mathcal{S}$. After reordering we may assume that $x^1, \ldots, x^k$ are precisely the patterns, containing $\tau$. The next calculation is similar to that of LP rounding analyses used in approximation algorithms for SetCover or MaxSat [Vaz01]. We bound the probability that task $\tau$ is not covered by any of the bought patterns as

$$\Pr\left[\bigwedge_{i=1}^{k} \text{pattern } x^i \text{ not bought}\right] \leq \prod_{i=1}^{k}(1 - \alpha\lambda_i) \leq e^{-\alpha \sum_{i=1}^{k} \lambda_i} \leq e^{-\alpha}$$

Here we use $1 - y \leq e^{-y}$ for all $y \in \mathbb{R}$ and $\lambda_1 + \ldots + \lambda_k \geq 1$. Let $\mathcal{S}'$ be the tasks, which are not yet covered, then $E[u(\mathcal{S}')] \leq e^{-\alpha} u(\mathcal{S})$. We distribute $\mathcal{S}$ via `FFMP` (see Algorithm 1 in Chapter 4) in a First Fit manner, then

$$E[\texttt{FFMP}(\mathcal{S}')] \leq 2 \cdot E[u(\mathcal{S}')] + 4 \leq 2e^{-\alpha} \cdot OPT^f_{\textsc{MulSched}}(\mathcal{S}) + 4$$

using Lemma 4.9 and $u(\mathcal{S}) \leq OPT^f_{\textsc{MulSched}}(\mathcal{S})$. Hence, the expected number of processors in the produced solution is at most

$$\alpha \cdot OPT^f_{\textsc{MulSched}}(\mathcal{S}) + 2e^{-\alpha} \cdot OPT^f_{\textsc{MulSched}}(\mathcal{S}) + 4$$

By the principle of the probabilistic method (see e.g. [AS08]), there must be at least one integral solution, which respects this bound. Choosing parameter $\alpha := \ln(2)$, we obtain

**Theorem 6.4.** *One has*

$$OPT_{\textsc{MulSched}}(\mathcal{S}) \leq (1 + \ln(2)) \cdot OPT^f_{\textsc{MulSched}}(\mathcal{S}) + 4,$$

*thus the asymptotic integrality gap of $(MLP)$ is upper bounded by $1 + \ln(2) < 1.694$.*

# Part II

# Intractability Results

# Chapter 7

# Diophantine Approximation

Diophantine approximation is one of the fundamental topics in mathematics. Roughly speaking, the objective is to replace a number or a vector, by another number or vector which is very close to the original, but less complex in terms of fractionality. A famous example is the Gregorian calendar, which approximates a solar year with its leap year rule.

Since the invention of the LLL algorithm by Lenstra, Lenstra & Lovász [LLL82], *simultaneous Diophantine approximation* has been a very important object of study also in computer science. One powerful result, for example, is the one of Frank & Tardos [FT87] who provided an algorithm based on Diophantine approximation and the LLL algorithm which, among other things, shows that a combinatorial 0/1-optimization problem is polynomial if and only if it is strongly polynomial.

Let us denote the distance of a real number $x \in \mathbb{R}$ to its nearest integer by $\{\{x\}\} = \min\{|x - z| \mid z \in \mathbb{Z}\}$. More general the distance of a vector $v \in \mathbb{R}^n$ to its nearest integer vector w.r.t. the $\| \cdot \|_\infty$-norm is $\{\{v\}\} = \min\{\|v - z\|_\infty \mid z \in \mathbb{Z}^n\}$.

Lagarias [Lag85] has shown that it is **NP**-complete to decide whether there exists an integer $Q \in \{1, \ldots, N\}$ with $\{\{Q \cdot \alpha\}\} \leq \varepsilon$, given $\alpha \in \mathbb{Q}^n$, $N \in \mathbb{N}$ and $\varepsilon > 0$. The *best approximation error* $\delta_N$ of a vector $\alpha \in \mathbb{Q}^n$ with denominator bound $N \in \mathbb{N}$ is defined as $\delta_N = \min\{\, \{\{Q \cdot \alpha\}\} \mid Q \in \{1, \ldots, N\}\,\}$. Lagarias [Lag85] showed also that the existence of a polynomial algorithm, which computes on input $\alpha \in \mathbb{Q}^n$ and $N \in \mathbb{N}$ a number $Q \in \{1, \ldots, 2^{n/2} \cdot N\}$ with $\{\{Q \cdot \alpha\}\} \leq \delta_N$ implies **NP** = **coNP**. The famous theorem of Dirichlet yields bounds, for which good approximations always exist.

**Theorem** (Dirichlet)**.** *For any $\alpha \in \mathbb{R}^n$ and $N \in \mathbb{N}$, there is a denominator $Q \in \{1, \ldots, N^n\}$ s.t. $\{\{Q\alpha\}\} \leq \frac{1}{N}$.*

Using the LLL algorithm [LLL82] one can at least find a $Q \leq 2^{\mathcal{O}(n^2)} N^n$, respecting the bound $\{\{Q\alpha\}\} \leq \frac{1}{N}$. Up to now there is no known polynomial time algorithm for determining a $Q$, as its existence is guaranteed by Dirichlet's theorem.

The reason for us to investigate simultaneous Diophantine approximation is, that it will yield a hardness proof for response time computation of implicit-deadline tasks as well as hardness of testing EDF-schedulability of constrained-deadline tasks and other problems. The key insight on the way to this result will be that simultaneous Diophantine approximation remains hard, if the approximation error is measured as distance to the nearest *larger* integer instead of the nearest one. The results, presented in this chapter, can be summarized in a simplified manner as

- *Simultaneous Diophantine approximation:* It is **NP**-hard to distinguish the cases
  - Yes : $\exists Q \in \{\lceil N/2 \rceil, \dots, N\} : \{\{Q\alpha\}\} \leq \varepsilon$
  - No : $\nexists Q \in \{1, \dots, 2^{n^{\mathcal{O}(1)}} N\} : \{\{Q\alpha\}\} \leq n^{\mathcal{O}(1/\log\log n)}\varepsilon$

  even if $\varepsilon \leq (\frac{1}{2})^{n^{\mathcal{O}(1)}}$.

- *Directed Diophantine approximation:* It is **NP**-hard to distinguish the cases
  - Yes : $\exists Q \in \{\lceil N/2 \rceil, \dots, N\} : \max_{i=1,\dots,n}(\lceil Q\alpha_i \rceil - Q\alpha_i) \leq \varepsilon$
  - No : $\nexists Q \in \{1, \dots, n^{\mathcal{O}(1/\log\log n)} N\} : \max_{i=1,\dots,n}(\lceil Q\alpha_i \rceil - Q\alpha_i) \leq 2^{n^{\mathcal{O}(1)}} \cdot \varepsilon$

  even if $\varepsilon \leq (\frac{1}{2})^{n^{\mathcal{O}(1)}}$.

- *Mixing Set:* Optimizing a linear function over a system

  $$\{(s, y) \in \mathbb{R}_{\geq 0} \times \mathbb{Z}^n \mid s + a_i\, y_i \geq b_i \ \forall i = 1, \dots, n\}$$

  is **NP**-hard. This answers an open question of Conforti et al. [CSW08].

- *Shortest vector in the positive orthant*: The problem of finding the shortest, nonnegative, nonzero vector in a lattice, i.e.

  $$\min \left\{ \|x\|_p \mid \exists z \in \mathbb{Z}^n : x = \sum_{i=1}^{n} z_i a_i, \ x \geq \mathbf{0}, \ x \neq \mathbf{0} \right\}$$

  is **NP**-hard to approximate within $n^{\mathcal{O}(1/\log\log n)}$ w.r.t. any $\|\cdot\|_p$-norm.

The promised hardness results for EDF-schedulability and response time computation will follow in Chapter 8. A complete overview over all reductions that we derive in this and in the next chapter is visualized in Figure 7.1. As a starting point, for all these results we first need a strengthening of the reduction of Lagarias.

## 7.1 Hardness of simultaneous Diophantine approximation

In complexity theory it is widely common to work with *gap problems* instead of the corresponding optimization variants, see for example the books of Vazirani [Vaz01] and Wegener [Weg05]. Since hardness results do not make sense for small instances, we will implicitly assume the phrase *for n large enough* in the upcoming reductions. In this section, we obtain that the following gap problem is **NP**-hard for $\rho_N = 2^{n^{c_1}}$ and $\rho_\varepsilon = n^{c_2/\log\log n}$ with $c_1, c_2 > 0$ being constants.

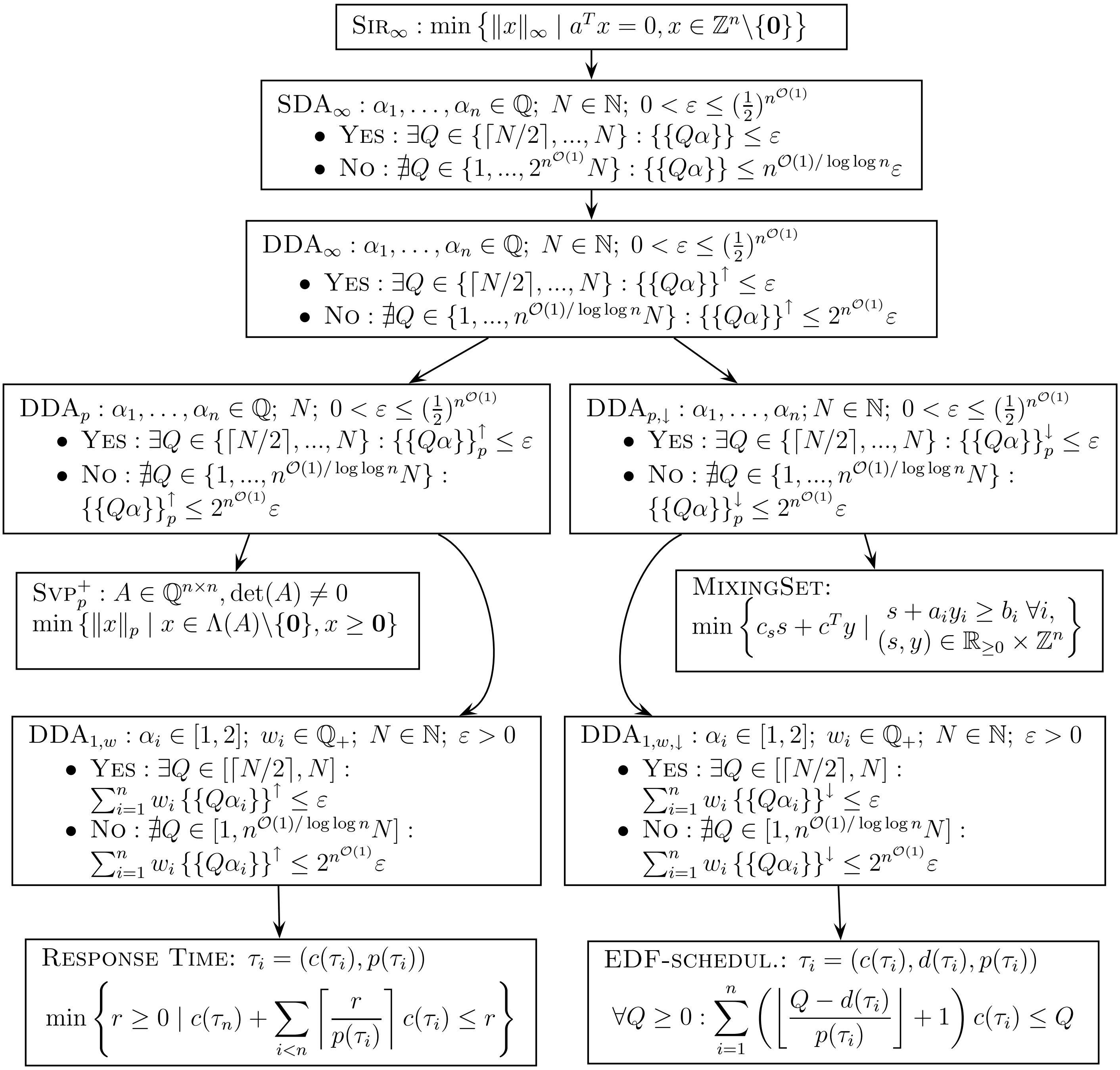

$\textsc{Sir}_\infty : \min\left\{\|x\|_\infty \mid a^Tx = 0, x \in \mathbb{Z}^n\setminus\{\mathbf{0}\}\right\}$
$\text{SDA}_\infty : \alpha_1,\dots,\alpha_n \in \mathbb{Q};\ N \in \mathbb{N};\ 0 < \varepsilon \le (\frac{1}{2})^{n^{\mathcal{O}(1)}}$
• $\textsc{Yes} : \exists Q \in \{\lceil N/2\rceil, ..., N\} : \{\{Q\alpha\}\} \le \varepsilon$
• $\textsc{No} : \nexists Q \in \{1, ..., 2^{n^{\mathcal{O}(1)}}N\} : \{\{Q\alpha\}\} \le n^{\mathcal{O}(1)/\log\log n}\varepsilon$
$\text{DDA}_\infty : \alpha_1,\dots,\alpha_n \in \mathbb{Q};\ N \in \mathbb{N};\ 0 < \varepsilon \le (\frac{1}{2})^{n^{\mathcal{O}(1)}}$
• $\textsc{Yes} : \exists Q \in \{\lceil N/2\rceil, ..., N\} : \{\{Q\alpha\}\}^\uparrow \le \varepsilon$
• $\textsc{No} : \nexists Q \in \{1, ..., n^{\mathcal{O}(1)/\log\log n}N\} : \{\{Q\alpha\}\}^\uparrow \le 2^{n^{\mathcal{O}(1)}}\varepsilon$
$\text{DDA}_p : \alpha_1,\dots,\alpha_n \in \mathbb{Q};\ N;\ 0 < \varepsilon \le (\frac{1}{2})^{n^{\mathcal{O}(1)}}$
• $\textsc{Yes} : \exists Q \in \{\lceil N/2\rceil, ..., N\} : \{\{Q\alpha\}\}_p^\uparrow \le \varepsilon$
• $\textsc{No} : \nexists Q \in \{1, ..., n^{\mathcal{O}(1)/\log\log n}N\} :$ $\{\{Q\alpha\}\}_p^\uparrow \le 2^{n^{\mathcal{O}(1)}}\varepsilon$
$\text{DDA}_{p,\downarrow} : \alpha_1,\dots,\alpha_n; N \in \mathbb{N};\ 0 < \varepsilon \le (\frac{1}{2})^{n^{\mathcal{O}(1)}}$
• $\textsc{Yes} : \exists Q \in \{\lceil N/2\rceil, ..., N\} : \{\{Q\alpha\}\}_p^\downarrow \le \varepsilon$
• $\textsc{No} : \nexists Q \in \{1, ..., n^{\mathcal{O}(1)/\log\log n}N\} :$ $\{\{Q\alpha\}\}_p^\downarrow \le 2^{n^{\mathcal{O}(1)}}\varepsilon$
$\textsc{Svp}_p^+ : A \in \mathbb{Q}^{n\times n}, \det(A) \neq 0$
$\min\left\{\|x\|_p \mid x \in \Lambda(A)\setminus\{\mathbf{0}\}, x \ge \mathbf{0}\right\}$
$\textsc{MixingSet}:$
$\min\left\{c_s s + c^T y \mid \begin{array}{c} s + a_i y_i \ge b_i\ \forall i, \\ (s,y) \in \mathbb{R}_{\ge 0} \times \mathbb{Z}^n \end{array}\right\}$
$\text{DDA}_{1,w} : \alpha_i \in [1,2];\ w_i \in \mathbb{Q}_+;\ N \in \mathbb{N};\ \varepsilon > 0$
• $\textsc{Yes} : \exists Q \in [\lceil N/2\rceil, N] :$ $\sum_{i=1}^n w_i \{\{Q\alpha_i\}\}^\uparrow \le \varepsilon$
• $\textsc{No} : \nexists Q \in [1, n^{\mathcal{O}(1)/\log\log n}N] :$ $\sum_{i=1}^n w_i \{\{Q\alpha_i\}\}^\uparrow \le 2^{n^{\mathcal{O}(1)}}\varepsilon$
$\text{DDA}_{1,w,\downarrow} : \alpha_i \in [1,2];\ w_i \in \mathbb{Q}_+;\ N \in \mathbb{N};\ \varepsilon > 0$
• $\textsc{Yes} : \exists Q \in [\lceil N/2\rceil, N] :$ $\sum_{i=1}^n w_i \{\{Q\alpha_i\}\}^\downarrow \le \varepsilon$
• $\textsc{No} : \nexists Q \in [1, n^{\mathcal{O}(1)/\log\log n}N] :$ $\sum_{i=1}^n w_i \{\{Q\alpha_i\}\}^\downarrow \le 2^{n^{\mathcal{O}(1)}}\varepsilon$
$\textsc{Response Time}: \tau_i = (c(\tau_i), p(\tau_i))$
$\min\left\{r \ge 0 \mid c(\tau_n) + \sum_{i<n} \left\lceil \frac{r}{p(\tau_i)} \right\rceil c(\tau_i) \le r\right\}$
$\text{EDF-}\textsc{schedul.}: \tau_i = (c(\tau_i), d(\tau_i), p(\tau_i))$
$\forall Q \ge 0 : \sum_{i=1}^n \left(\left\lfloor \frac{Q - d(\tau_i)}{p(\tau_i)} \right\rfloor + 1\right) c(\tau_i) \le Q$

Figure 7.1: Overview over reductions. Arrows indicate Karp reductions. $\mathcal{O}(1)$ stands for either for the phrase *for all constants* or for *some constant*. We abbreviate $\{\{x\}\}_p^\uparrow := (\sum_{i=1}^n (\lceil x \rceil - x)^p)^{1/p}$, $\{\{x\}\}_p^\downarrow := (\sum_{i=1}^n (x - \lfloor x \rfloor)^p)^{1/p}$, $\{\{x\}\}^\uparrow := \{\{x\}\}_\infty^\uparrow$ and $\{\{x\}\}^\downarrow := \{\{x\}\}_\infty^\downarrow$.

Simultaneous Diophantine Approximation ($\mathrm{SDA}_\infty^{\rho_N,\rho_\varepsilon}$)
Given: $\alpha_1,\dots,\alpha_n \in \mathbb{Q}$; $N \in \mathbb{N}$; $0 < \varepsilon \leq 1/\rho_N^2$
Distinguish:

- Yes : $\exists Q \in \{\lceil N/2 \rceil, \dots, N\} : \{\{Q\alpha\}\} \leq \varepsilon$
- No : $\nexists Q \in \{1, \dots, \rho_N \cdot N\} : \{\{Q\alpha\}\} \leq \rho_\varepsilon \cdot \varepsilon$

In other words, we will prove that there is a polynomial time algorithm, taking the instance of an **NP**-complete problem, say a clause $C$ for Sat, and yielding a $\mathrm{SDA}_\infty^{\rho_N,\rho_\varepsilon}$ instance, such that we are in the Yes case if $C$ is satisfiable and we are in the No case otherwise. Thus we will never be obliged to deal with $\mathrm{SDA}_\infty^{\rho_N,\rho_\varepsilon}$ instances that fall in neither of these two cases. Note that usually gap problems are equipped with just one parameter, not with two like in case of $\mathrm{SDA}_\infty^{\rho_N,\rho_\varepsilon}$. In the definition of $\mathrm{SDA}_\infty^{\rho_N,\rho_\varepsilon}$, one has the constraint that $\varepsilon$ must be very small, namely $\varepsilon \leq 1/\rho_N^2$. In fact, the hardness reduction will yield instances, respecting this condition and we will need such an assumption on $\varepsilon$ later on. We will always assume $\rho_\varepsilon$ and $\rho_N$ to be integers and $\rho_\varepsilon, \rho_N \geq 2$. If we are discussing simultaneous Diophantine approximation without referring to particular gap parameters, we will denote this simply by $\mathrm{SDA}_\infty$.

Let $\mathrm{Sir}_\infty$ be the problem of finding a shortest nontrivial integer vector in a hyperplane.

Shortest Integer Relation ($\mathrm{Sir}_\infty$)
Given: $a \in \mathbb{Z}^n$
Find: $OPT_{\mathrm{Sir}_\infty}(a) = \min\{\|x\|_\infty \mid a^T x = 0, x \in \mathbb{Z}^n \backslash \{\mathbf{0}\}\}$

Dinur [Din02] gave a reduction from Super-Sat to the shortest vector problem in the infinity norm. This reduction has been modified by Chen & Meng [CM07] to obtain

**Theorem 7.1.** *[CM07] For a universal constant $c > 0$, given a vector $a \in \mathbb{Z}^n \backslash \{\mathbf{0}\}$ it is* **NP**-*hard to distinguish*

- Yes : $OPT_{\mathrm{Sir}_\infty} = 1$
- No : $OPT_{\mathrm{Sir}_\infty} \geq n^{c/\log\log n}$.

We will revisit the reduction from $\mathrm{Sir}_\infty$ to simultaneous Diophantine approximation of Lagarias [Lag85] and Rössner & Seifert [RS96] to obtain

**Theorem 7.2** (Hardness of $\mathrm{SDA}_\infty^{\rho_N,\rho_\varepsilon}$)**.** *There is a universal constant $c_2 > 0$, such that for all fixed $c_1 > 0$, $\mathrm{SDA}_\infty^{\rho_N,\rho_\varepsilon}$ is* **NP**-*hard for $\rho_N = 2^{n^{c_1}}$ and $\rho_\varepsilon = n^{c_2/\log\log n}$, i.e. for rational numbers $(\alpha_1, \dots, \alpha_n)$, a bound $N \in \mathbb{N}$ and an error bound $0 < \varepsilon \leq 2^{-2n^{c_1}}$ it is* **NP**-*hard to distinguish the following cases*

- Yes : $\exists Q \in \{\lceil N/2 \rceil, \dots, N\} : \{\{Q\alpha\}\} \leq \varepsilon$
- No : $\nexists Q \in \{1, \dots, 2^{n^{c_1}} N\} : \{\{Q\alpha\}\} \leq n^{c_2/\log\log n}\varepsilon$.

This is a strengthening of the known reduction, which yields the following improvements:

1. In the Yes case, there exists a good $Q$ which is at least $\lceil N/2 \rceil$ whereas the result in [RS96] guarantees only a good $Q$ in the interval $\{1, \dots, N\}$.

2. In the No case, each $Q$ bounded by $2^{n^{c_1}} \cdot N$ is violating the distance bound, whereas the reduction of [RS96] together with the result of [CM07] guarantees this violation only for $Q \in \{1, \dots, \lfloor n^{c_2/\log\log n} \rfloor \cdot N\}$.

3. The bound of $\varepsilon \leq (\frac{1}{2})^{2n^{c_1}}$ was not part of the original reduction [Lag85, RS96].

These improvements will turn out to be crucial for our further hardness results in which $\text{SDA}_\infty^{\rho_N, \rho_\varepsilon}$ is used as a starting point.

## Assumptions on $\text{SIR}_\infty$

A solution $x$ to $\text{SIR}_\infty$ must be nontrivial, thus there must be an index $i$ with $x_i \neq 0$. Since $-x$ is a solution as well, we may assume that $x_i \geq 1$. However in the reduction from $\text{SIR}_\infty$ to $\text{SDA}_\infty^{\rho_N, \rho_\varepsilon}$ we will suppose that in fact $i = 1$. Let us first discuss, why this assumption may be made without loss of generality.

To this end, consider the matrix

$$A = \begin{pmatrix} 0 & a^T & \mathbf{0}^T & \dots & \mathbf{0}^T \\ 0 & \mathbf{0}^T & a^T & \dots & \mathbf{0}^T \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \mathbf{0}^T & \mathbf{0}^T & \dots & a^T \\ -1 & e_1^T & e_2^T & \dots & e_n^T \end{pmatrix} \in \mathbb{Z}^{(n+1)\times(n^2+1)}$$

containing $n$ copies of $a^T$ on a shifted diagonal and having the last row $(-1, e_1^T, e_2^T, \dots, e_n^T)$, where $e_i$ is the $i$-th $n$-dimensional unit column vector. The rest is filled by zeros.

**Lemma 7.3.** *One has*

$$OPT_{\text{SIR}_\infty}(a) = \min\{\|x\|_\infty \mid Ax = \mathbf{0}, x \in \mathbb{Z}^{n^2+1} \backslash \{\mathbf{0}\}\}$$

*and there is always an optimum solution $x$ for the right hand side expression with $x_1 \geq 1$.*

*Proof.* Let $x$ be a $\text{SIR}_\infty(a)$ solution, thus $a^T x = \mathbf{0}$ and $x_i \neq 0$ for some $i$. After possibly switching to $-x$ we may assume that $x_i \geq 1$. Then for

$$y := (x_i, \underbrace{\mathbf{0}, \dots, \mathbf{0}}_{i-1 \text{ times}}, x, \underbrace{\mathbf{0}, \dots, \mathbf{0}}_{n-i \text{ times}})$$

we have $Ay = \mathbf{0}$ and $\|y\|_\infty = \|x\|_\infty$.

Now let vice versa $y = (z, x^1, \dots, x^n)$ be a non-trivial integer solution with $Ay = \mathbf{0}$. Then $a^T x^i = 0$ for all $i = 1, \dots, n$. The only possibility, that no $x^i$ is a suitable $\text{SIR}_\infty$ solution for $a$ would be that $x^1 = \dots = x^n = \mathbf{0}$. But then $z = 0$, contradicting that $y \neq \mathbf{0}$. □

Kannan [Kan83] provided an algorithm, replacing a system $Ax = \mathbf{0}$ by one equation $a'^T x = 0$ in polynomial time such that

$$\{x \in \mathbb{Z}^{n^2+1} \mid Ax = \mathbf{0}, \|x\|_\infty \leq \mu\} = \{x \in \mathbb{Z}^{n^2+1} \mid a'^T x = 0, \|x\|_\infty \leq \mu\}.$$

His algorithm is polynomial in the encoding length of $A$ and $\mu$. Choosing $\mu = n$ is enough for our purposes so that Kannan's algorithm yields the desired shortest integer relation instance, where one always has optimum solutions whose first entry is positive.

## The reduction from $\textsc{Sir}_\infty$ to $\mathbf{SDA}_\infty^{\rho_N,\rho_\varepsilon}$

We are now prepared for the reduction.

**Theorem 7.4.** *Let $\rho_\varepsilon \leq n$ and $2^n \leq \rho_N = 2^{n^{\mathcal{O}(1)}}$. There is a polynomial time transformation, which takes an $\textsc{Sir}_\infty$ instance $a \in \mathbb{Z}^n$ and yields a $\text{SDA}_\infty^{\rho_N,\rho_\varepsilon}$ instance consisting of rational numbers $(\alpha_0, \dots, \alpha_n)$, a bound $N \in \mathbb{N}$ and an error bound $0 < \varepsilon \leq 1/\rho_N^2$ such that*

- $OPT_{\textsc{Sir}_\infty} = 1 \Rightarrow \exists Q \in \{\lceil N/2 \rceil, \dots, N\} : \{\{Q\alpha\}\} \leq \varepsilon$
- $OPT_{\textsc{Sir}_\infty} > \rho_\varepsilon \Rightarrow \nexists Q \in \{1, \dots, \rho_N \cdot N\} : \{\{Q\alpha\}\} \leq \rho_\varepsilon \cdot \varepsilon$

Observe that the gap parameter $\rho_\varepsilon$ is conveyed from $\textsc{Sir}_\infty$ to $\text{SDA}_\infty^{\rho_N,\rho_\varepsilon}$, while the second parameter $\rho_N$ can be chosen arbitrarily large, as long as its encoding length is polynomial. Before we state the instance $(\alpha_0, \dots, \alpha_n; N; \varepsilon)$ for the reduction, we need to determine suitable prime numbers. One can find different primes $p, q_1, \dots, q_n$ as well as natural numbers $R$ and $T$ in polynomial time, such that

1. $A := n \sum_{j=1}^n |a_j| < p^R < q_1^T < q_2^T < \dots < q_n^T < (1 + \frac{1}{n}) \cdot q_1^T$
2. $p$ and all $q_i$ are co-prime to all $a_j$
3. $q_1^T > \rho_N^2 \cdot p^R$
4. $T, R, p, q_1, \dots, q_n$ are bounded by a polynomial in the input length.

A proof of this claim with weaker bounds is presented in [Lag85, RS96]. A full proof can be found in the Appendix for the sake of completeness.

The following system of congruences appears already in a paper of Adleman & Manders [MA78] and is crucial for the reduction.

$$\begin{aligned} r_j &\equiv_{q_i^T} 0 \ \forall i \neq j && (7.1)\\ r_j &\equiv_{p^R} a_j && (7.2)\\ r_j &\not\equiv_{q_j} 0 && (7.3) \end{aligned}$$

Since the moduli $q_i^T, p^R$ are co-prime there are solutions for $r_j$, by the *Chinese Remainder Theorem*, see e.g. [NZM91]. For $j = 2, \dots, n$, choose the smallest possible solution for $r_j$. There is also an efficient way to compute $r_j$. Define $B := \prod_{j=1}^n q_j^T$, then the Chinese Remainder Theorem allows to compute some $r_j' \leq p^R B / q_j^T$, which simultaneously solve

(7.1) and (7.2). Then there are two possibilities: Either we have $r'_j \not\equiv_{q_j} 0$, then $r_j := r'_j$ is a suitable choice. Otherwise we have $r_j := r'_j + p^R B/q_j^T \not\equiv_{q_j} 0$, while (7.1) and (7.2) still hold. In any case $r_j \leq 2p^R B/q_j^T$.

We want to choose $r_1$ considerably larger than $r_2, \ldots, r_n$. Namely $r_1 = r'_1 + 12np^R B/q_1^T$ or $r_1 = r'_1 + (12n+1)p^R B/q_1^T$; at least one choice solves (7.1), (7.2) & (7.3). In any case $12np^R B/q_1^T \leq r_1 \leq 13np^R B/q_1^T$ and $r_1 \geq 6n \cdot r_j$ for all $j = 2, \ldots, n$. The reason for the special treatment of $r_1$ is that we assume $x_1 \geq 1$. This will later cause that $Q \geq N/2$ in the YES case.

But we first need the following observation

**Lemma 7.5.** *The systems*

$$\underbrace{\begin{array}{r} \sum_{j=1}^{n} x_j a_j = 0 \\ x \in \mathbb{Z}^n \\ 1 \leq \|x\|_\infty \leq n \end{array}}_{(I)} \quad \textit{and} \quad \underbrace{\begin{array}{l} \sum_{j=1}^{n} x_j r_j \equiv_{p^R} 0 \\ x \in \mathbb{Z}^n \\ 1 \leq \|x\|_\infty \leq n \end{array}}_{(II)}$$

*have the same set of solutions.*

*Proof.* Since $a_j \equiv_{p^R} r_j$, each solution $x$ for $(I)$ is a solution for $(II)$. Vice versa, let $x$ be a solution for $(II)$, thus $\sum_{j=1}^{n} x_j r_j \equiv_{p^R} 0$. Due to $a_j \equiv_{p^R} r_j$, the congruence $\sum_{j=1}^{n} x_j a_j \equiv_{p^R} 0$ holds. But we have

$$|\sum_{j=1}^{n} x_j a_j| \leq \underbrace{\underbrace{\|x\|_\infty}_{\leq n} \cdot \sum_{j=1}^{n} |a_j|}_{\leq A} < p^R$$

thus $\sum_{j=1}^{n} x_j a_j = 0$. We conclude that $x$ solves $(I)$. □

Basically this lemma allows us, to replace each $a_j$ by a value $r_j$, having additional properties. This procedure will pay off later.

By $r_j^{-1} \in \mathbb{Z}_{q_j^T}$ we denote the unique value s.t. $r_j \cdot r_j^{-1} \equiv_{q_j^T} 1$ (this must exist since $r_j \not\equiv_{q_j} 0$ implies that $\gcd(r_j, q_j^T) = 1$ and consequently $r_j$ is a unit in the ring $\mathbb{Z}_{q_j^T}$). The $\mathrm{SDA}_\infty^{\rho_N,\rho_\varepsilon}$-instance for the reduction is

$$\begin{aligned} \alpha_0 &:= \frac{1}{p^R} \\ \alpha_j &:= \frac{r_j^{-1}}{q_j^T} \quad \forall j = 1, \ldots, n \\ N &:= \sum_{j=1}^{n} r_j \\ \varepsilon &:= \frac{1}{q_1^T}. \end{aligned}$$

Note that $\varepsilon = \frac{1}{q_1^T} \leq \frac{1}{\rho_N^2}$ by the choice of $q_1, \dots, q_n$, thus the bound on $\varepsilon$ is justified. To give some intuition behind this system: Since all $q_j^T$ are co-prime, there is a one-to-one correspondence between solutions $x$ and good Diophantine approximations $Q$. We will see that $x$ lies on the hyperplane $a^T x = 0$ if and only if the corresponding $Q$ is a multiple of $p^R$ (at least for the relevant $x$ with $\|x\|_\infty \leq n$). Moreover the distance of $Q\alpha_j$ to the next integer will be proportional to $x_j$.

**Theorem 7.6.** *If $OPT_{\text{SIR}_\infty} = 1$, then there is a $Q \in \{\lceil N/2 \rceil, \dots, N\}$ with $\{\{Q\alpha\}\} \leq \varepsilon$.*

*Proof.* Let $x \in \{0, \pm 1\}^n \backslash \{\mathbf{0}\}$ be that $\text{SIR}_\infty$ solution. We already outlined that we may assume $x_1 = 1$. Choose $Q := \sum_{j=1}^n r_j x_j$ as $\text{SDA}_\infty$ solution. Clearly one has

$$Q = \sum_{j=1}^n \underbrace{x_j}_{\leq 1} \cdot r_j \leq N.$$

On the other hand applying $x_1 = 1$ yields

$$Q \geq r_1 - \sum_{j=2}^n \underbrace{r_j}_{\leq r_1/(6n)} \geq \frac{1}{2} \sum_{j=1}^n r_j = N/2,$$

thus $Q$ is within the feasible bounds. It remains to show that $Q$ gives a good approximation. Note that

$$\{\{Q\alpha_0\}\} = \left\{\left\{ \frac{\sum_{j=1}^n x_j r_j}{p^R} \right\}\right\} = 0,$$

due to the reason that $a^T x = 0$ and therefore $\sum_{j=1}^n r_j x_j \equiv_{p^R} 0$ (see Lemma 7.5). Furthermore we derive that for $i \in \{1, \dots, n\}$ one has

$$\{\{Q\alpha_i\}\} = \left\{\left\{ r_i^{-1} \frac{\sum_{j=1}^n x_j \cdot r_j}{q_i^T} \right\}\right\} = \left\{\left\{ \frac{x_i r_i r_i^{-1}}{q_i^T} \right\}\right\} \leq \frac{1}{q_i^T} \leq \frac{1}{q_1^T} = \varepsilon$$

using that $r_j \equiv_{q_i^T} 0$ for $i \neq j$ and $r_i \cdot r_i^{-1} \equiv_{q_i^T} 1$. The claim then follows. □

Next, we show the reverse direction of the reduction.

**Theorem 7.7.** *Given a* $\text{SDA}_\infty$ *solution $Q \in \{1, \dots, \rho_N \cdot N\}$ with $\{\{Q\alpha\}\} \leq \rho_\varepsilon \cdot \varepsilon$, then $OPT_{\text{SIR}_\infty} \leq \rho_\varepsilon$.*

*Proof.* We construct a solution $\hat{x}$ for our $\text{SIR}_\infty$ instance $a^T x = 0$ as follows: Let $\hat{x}_j$ be the smallest integer in absolute value with

$$Q r_j^{-1} \equiv_{q_j^T} \hat{x}_j.$$

For proving that $\hat{x}$ is the desired $\text{SIR}_\infty$ solution we need to show three partial claims, namely

$$\|\hat{x}\|_\infty \leq \rho_\varepsilon, \quad \hat{x} \neq \mathbf{0} \quad \text{and} \quad a^T \hat{x} = 0. \tag{7.4}$$

The first assertion of (7.4) follows from the fact that $q_1^T < q_j^T < (1+1/n)\cdot q_1^T \leq (1+1/\rho_\varepsilon)\cdot q_1^T$ which implies the strict inequality in

$$\left|\frac{\hat{x}_j}{q_j^T}\right| = \left\{\left\{\frac{Qr_j^{-1}}{q_j^T}\right\}\right\} \leq \rho_\varepsilon \cdot \varepsilon = \frac{\rho_\varepsilon}{q_1^T} < \frac{\rho_\varepsilon + 1}{q_j^T}.$$

Observe that $Q$ is a multiple of $p^R$. If this was not the case, then

$$\{\{Q\alpha_0\}\} = \left\{\left\{\frac{Q}{p^R}\right\}\right\} \geq \frac{1}{p^R} > \frac{\rho_\varepsilon}{q_1^T} = \rho_\varepsilon \cdot \varepsilon,$$

since $q_1^T > \rho_N^2 p^R$ and $\rho_\varepsilon \leq n \leq \rho_N$.

We next show that $Q = \sum_{i=1}^n \hat{x}_i r_i$. This implies directly that $\hat{x} \neq \mathbf{0}$, since $Q \geq 1$. Furthermore $Q \equiv_{p^R} 0$ and Lemma 7.5 imply together with $\|\hat{x}\|_\infty \leq \rho_\varepsilon \leq n$ that $a^T\hat{x} = 0$ and (7.4) is proved.

Multiplying the equation $Q \cdot r_j^{-1} \equiv_{q_j^T} \hat{x}_j$ with $r_j$ yields $Q \equiv_{q_j^T} \hat{x}_j r_j$. Recall that $B = \prod_{j=1}^n q_j^T$. Then the following implication holds

$$\begin{bmatrix} Q & \equiv_{q_j^T} & \hat{x}_j r_j & \forall j \\ 0 & \equiv_{q_i^T} & \hat{x}_j r_j & \forall i \neq j \end{bmatrix} \Rightarrow \left[Q \equiv_B \sum_{j=1}^n \hat{x}_j r_j\right]$$

since the left hand side equations guarantee that $Q \equiv_{q_i^T} \sum_{j=1}^n \hat{x}_j r_j$ for all divisors $q_i^T$ of $B$.

We are done with the proof, once we have shown that $Q < B/2$ and $|\sum_{j=1}^n \hat{x}_j r_j| < B/2$, since then both values must coincide, if they are congruent to each other modulo $B$.

We first bound the value of $N$. This is done by applying the bound $r_j \leq 13 \cdot np^R \cdot B/q_1^T$. We obtain

$$N = \sum_{j=1}^n r_j \leq 13n^2 \frac{p^R}{q_1^T} \cdot B < \frac{1}{2} \cdot \frac{B}{\rho_N}$$

for $n$ large enough (recall that $\rho_N \geq 2^n$ and $q_1^T > \rho_N^2 \cdot p^R$). Finally $Q \leq \rho_N \cdot N < \frac{B}{2}$ and

$$|\sum_{j=1}^n \hat{x}_j r_j| \leq \underbrace{\max_{j=1,\dots,n} |\hat{x}_j|}_{\leq \rho_\varepsilon} \cdot \underbrace{\sum_{j=1}^n r_j}_{=N} \leq \rho_\varepsilon \cdot N \leq \rho_N \cdot N < \frac{B}{2}$$

concluding the claim. □

Combining the results, Theorem 7.4 follows.

*Remark* 7.8. Note that the restriction $\rho_\varepsilon \leq n$ could be replaced by any polynomial upper bound on $\rho_\varepsilon$. Furthermore the YES case can be modified such that not just $Q \geq \lceil N/2 \rceil$, but even $Q \geq \lceil N/(1+1/f(m)) \rceil$ and $\varepsilon \leq 1/f(m)$, where $f$ is an arbitrary polynomial time computable (positive) function and $m$ is the input length of the instance.

## 7.2 Directed Diophantine approximation

In a next step, we want to prove that the hardness of simultaneous Diophantine approximation still holds if instead of rounding to the next integer, we measure the approximation error as the distance to the next larger integer. We term this problem *directed Diophantine approximation.* In case that $x \in \mathbb{R}$ is a number, define $\lceil x \rceil = \min\{z \in \mathbb{Z} \mid z \geq x\}$ and $\{\{x\}\}^{\uparrow} = \lceil x \rceil - x$. For a vector $v \in \mathbb{R}^n$ we denote $\{\{v\}\}^{\uparrow} = \max_{i=1,\dots,n}(\lceil v_i \rceil - v_i)$. Similar we define $\{\{x\}\}^{\downarrow} = x - \lfloor x \rfloor$ as the distance to the next smaller integer. Observe that $\{\{x\}\}^{\uparrow} = \{\{-x\}\}^{\downarrow}$.

The gap problem for which we derive hardness in this section is formally defined as

Directed Diophantine Approximation ($\text{DDA}_{\infty}^{\rho_N,\rho_\varepsilon}$)
Given: $\alpha_1, \dots, \alpha_n \in \mathbb{Q},\ N \in \mathbb{N},\ 0 < \varepsilon \leq 3/\rho_\varepsilon^2$
Distinguish:

- Yes : $\exists Q \in \{\lceil N/2 \rceil, \dots, N\} : \{\{Q\alpha\}\}^{\uparrow} \leq \varepsilon$
- No : $\nexists Q \in \{1, \dots, \rho_N \cdot N\} : \{\{Q\alpha\}\}^{\uparrow} \leq \rho_\varepsilon \cdot \varepsilon$

Directed Diophantine approximation was for example considered by Henk & Weismantel [HW97, HW02] in the context of an integer programming problem, while its computational complexity remained open.

The proof idea for our reduction from $\text{SDA}_\infty$ to $\text{DDA}_\infty$ works as follows: For each rational $\alpha_i$ in the $\text{SDA}_\infty$ instance, we add numbers $\alpha_i - \delta$ and $-(\alpha_i + \delta)$ to the $\text{DDA}_\infty$ instance. Next consider a number $Q$ such that $Q\alpha_i$ is close to an integer, say $z_i$. Then for a suitable choice of $\delta$, $Q\alpha_i - Q\delta$ is slightly smaller than $z_i$ and $Q\alpha_i + Q\delta$ is slightly larger. Hence, $\{\{Q\alpha_i - Q\delta\}\}^{\uparrow}$ and $\{\{Q\alpha_i + Q\delta\}\}^{\downarrow} = \{\{Q \cdot (-(\alpha_i + \delta))\}\}^{\uparrow}$ are both small, since $Q(\alpha_i \pm \delta)$ is still close to $z_i$. Vice versa, suppose $\{\{Q\alpha_i - Q\delta\}\}^{\uparrow}$ and $\{\{Q\alpha_i + Q\delta\}\}^{\downarrow}$ are both small. We will then see that $\lceil Q\alpha_i - Q\delta \rceil = \lfloor Q\alpha_i + Q\delta \rfloor =: z_i$ and consequently $Q\alpha_i$ is close to the integer $z_i$.

Note that in the following reduction, the gap parameters $\rho_\varepsilon$ and $\rho_N$ will change their position.

**Theorem 7.9.** *Let $\rho_N, \rho_\varepsilon > 6$. There is a polynomial time reduction taking a* $\text{SDA}_\infty$ *instance $(\alpha_1, \dots, \alpha_n; N; \varepsilon)$ with $\varepsilon \leq 1/\rho_\varepsilon^2$ as input and producing a* $\text{DDA}_\infty$ *instance $(\alpha'_1, \dots, \alpha'_{2n}; N; \varepsilon')$ such that $\varepsilon' \leq 3/\rho_\varepsilon^2$ and*

- Yes : $\exists Q \in \{\lceil N/2 \rceil, \dots, N\} : \{\{Q\alpha\}\} \leq \varepsilon$
  $\Rightarrow \exists Q \in \{\lceil N/2 \rceil, \dots, N\} : \{\{Q\alpha'\}\}^{\uparrow} \leq \varepsilon'$
- No : $\nexists Q \in \{1, \dots, \rho_N \cdot N\} : \{\{Q\alpha\}\} \leq 2\rho_N \cdot \varepsilon$
  $\Rightarrow \nexists Q \in \{1, \dots, \rho_N \cdot N\} : \{\{Q\alpha'\}\}^{\uparrow} \leq \rho_\varepsilon \cdot \varepsilon'$

*Proof.* Let $\alpha_1, \dots, \alpha_n$ together with $N \in \mathbb{N}$ and $\varepsilon$ be the given $\text{SDA}_\infty$ instance. We define a $\text{DDA}_\infty$ instance as

$$\begin{aligned} \alpha'_i &:= \alpha_i - \delta \quad \forall i = 1, \dots, n \\ \alpha'_{i+n} &:= -(\alpha_i + \delta) \quad \forall i = 1, \dots, n \\ \varepsilon' &:= 3\varepsilon \end{aligned}$$

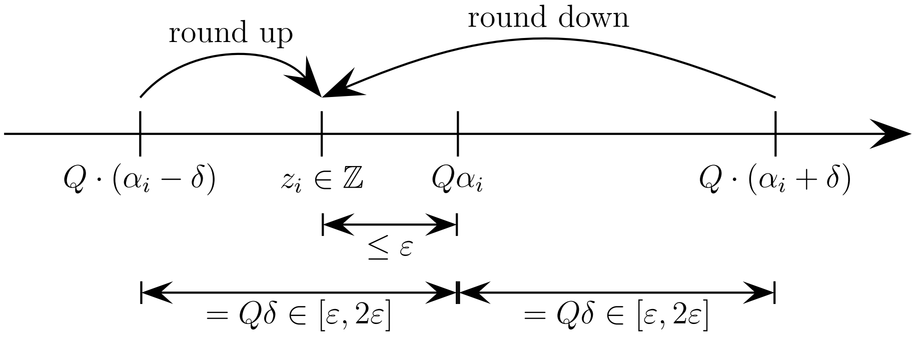
with $\delta := \frac{2\varepsilon}{N}$. We start by proving the first implication. Let $Q \in \{\lceil N/2 \rceil, \dots, N\}$ be a $\mathrm{SDA}_\infty$ solution, i.e.

$$\max_{i=1,\dots,n} |\lceil Q\alpha_i \rfloor - Q\alpha_i| \leq \varepsilon.$$

Choose an arbitrary $i \in \{1, \dots, n\}$ and let $z_i$ be the integer, such that $|Q\alpha_i - z_i| \leq \varepsilon$. Since $Q \cdot \delta \geq \frac{N}{2} \cdot \frac{2\varepsilon}{N} = \varepsilon$, we have $Q(\alpha_i + \delta) \geq z_i$. We conclude that

$$\{\{Q \cdot (-(\alpha_i + \delta))\}\}^{\uparrow} = \{\{Q(\alpha_i + \delta)\}\}^{\downarrow} \leq \underbrace{|Q\alpha_i - z_i|}_{\leq \varepsilon} + \underbrace{Q\delta}_{\leq N \cdot \frac{2\varepsilon}{N} = 2\varepsilon} \leq 3\varepsilon$$

This situation is depicted below:

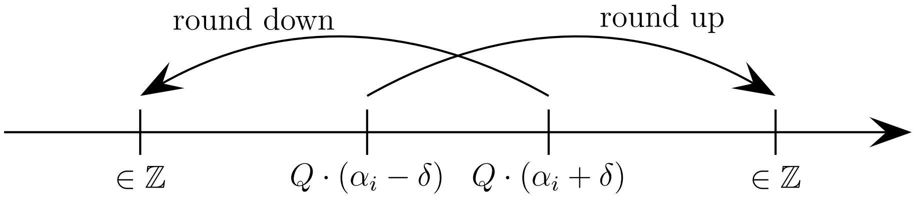

round up
round down
$Q \cdot (\alpha_i - \delta)$
$z_i \in \mathbb{Z}$
$Q\alpha_i$
$Q \cdot (\alpha_i + \delta)$
$\leq \varepsilon$
$= Q\delta \in [\varepsilon, 2\varepsilon]$
$= Q\delta \in [\varepsilon, 2\varepsilon]$

Similar one has

$$\{\{Q(\alpha_i - \delta)\}\}^{\uparrow} \leq 3\varepsilon$$

for all $i = 1, \dots, n$, thus $\{\{Q\alpha'\}\}^{\uparrow} \leq 3\varepsilon = \varepsilon'$ and the first implication is proven.

It remains to show the second part of the reduction, namely that given a $Q \in \{1, \dots, \rho_N \cdot N\}$ with $\{\{Q\alpha'\}\}^{\uparrow} \leq \rho_\varepsilon \cdot \varepsilon'$ one also has $\{\{Q\alpha\}\} \leq 2\rho_N \cdot \varepsilon$. Assume for contradiction that there is an $i \in \{1, \dots, n\}$, such that there is no integer $z_i$ lying between $Q(\alpha_i - \delta)$ and $Q(\alpha_i + \delta)$ (see the following figure)



But then either $\{\{Q(\alpha_i - \delta)\}\}^{\uparrow} \geq 1/2$ or $\{\{Q(\alpha_i + \delta)\}\}^{\downarrow} \geq 1/2$. In any case $\{\{Q\alpha'\}\}^{\uparrow} \geq 1/2$, contradicting that $\{\{Q\alpha'\}\}^{\uparrow} \leq \rho_\varepsilon \cdot \varepsilon' = \rho_\varepsilon \cdot 3\varepsilon < 1/2$, where we use the assumption $\varepsilon \leq 1/\rho_\varepsilon^2$. Now let $z_i \in \mathbb{Z}$ be the integer such that

$$Q(\alpha_i - \delta) \leq z_i \leq Q(\alpha_i + \delta).$$

Then

$$|Q\alpha_i - z_i| \leq \delta \cdot Q \leq \frac{2\varepsilon}{N} \cdot \rho_N N = 2\rho_N \cdot \varepsilon$$

proving the claim. □

Combining Theorem 7.9 with the hardness result of $\mathrm{SDA}_\infty^{\rho_N,\rho_\varepsilon}$ and the fact that $2n^{c_2/\log\log n} \leq n^{c_2'/\log\log n}$ for $c_2' := c_2/2$ and $n$ large enough, one obtains:

**Theorem 7.10** (Hardness of $\mathrm{DDA}_\infty^{\rho_N,\rho_\varepsilon}$)**.** *There is a universal constant $c_2 > 0$ such that for all fixed $c_1 > 0$, $\mathrm{DDA}_\infty^{\rho_N,\rho_\varepsilon}$ is* **NP***-hard for $\rho_\varepsilon = 2^{n^{c_1}}$ and $\rho_N = \lfloor n^{c_2/\log\log n} \rfloor$, i.e. for rational numbers $(\alpha_1, \dots, \alpha_n)$, bounds $N \in \mathbb{N}$ and $0 < \varepsilon \leq 3/\rho_\varepsilon^2$ it is* **NP***-hard to distinguish the following cases*

- Yes : $\exists Q \in \{\lceil N/2 \rceil, \dots, N\} : \{\{Q\alpha\}\}^\uparrow \leq \varepsilon$
- No : $\nexists Q \in \{1, \dots, \lfloor n^{c_2/\log\log n} \rfloor \cdot N\} : \{\{Q\alpha\}\}^\uparrow \leq 2^{n^{c_1}}\varepsilon.$

We conjecture that $\mathrm{DDA}_\infty$ is in fact much harder than we were able to prove:

**Conjecture 7.11.** $\mathrm{DDA}_\infty^{\rho_N,\rho_\varepsilon}$ *is* **NP***-hard for $\rho_N = \rho_\varepsilon = 2^{n^c}$ and any constant $c > 0$.*

Till now we measured the approximation error with the $\|\cdot\|_\infty$-norm, but of course we can also use any other $\|\cdot\|_p$-norm. Especially the $\|\cdot\|_1$-norm will turn out to be a more natural starting point for the upcoming reductions.

Directed Diophantine Approximation w.r.t. $\|\cdot\|_p$-norm ($\mathrm{DDA}_p^{\rho_N,\rho_\varepsilon}$)
Given: $\alpha_1, \dots, \alpha_n \in \mathbb{Q}$, $N \in \mathbb{N}$, $0 < \varepsilon \leq 3/\rho_\varepsilon^2$
Distinguish:

- Yes : $\exists Q \in \{\lceil N/2 \rceil, \dots, N\} : \left(\sum_{i=1}^n |\lceil Q\alpha_i \rceil - Q\alpha_i|^p\right)^{1/p} \leq \varepsilon$
- No : $\nexists Q \in \{1, \dots, \rho_N \cdot N\} : \left(\sum_{i=1}^n |\lceil Q\alpha_i \rceil - Q\alpha_i|^p\right)^{1/p} \leq \rho_\varepsilon \cdot \varepsilon$

Since all $\|\cdot\|_p$-norms differ by at most a factor of $n$ and the hardness gap $\rho_\varepsilon$ is of the form $2^{n^{c_1}} \gg n$, adapting the constant $c_1$ yields

**Corollary 7.12** (Hardness of $\mathrm{DDA}_p^{\rho_N,\rho_\varepsilon}$)**.** *There is a universal constant $c_2 > 0$ such that for all fixed $c_1 > 0$ and all $1 \leq p \leq \infty$, $\mathrm{DDA}_p^{\rho_N,\rho_\varepsilon}$ is* **NP***-hard for $\rho_\varepsilon = 2^{n^{c_1}}$ and $\rho_N = \lfloor n^{c_2/\log\log n} \rfloor$.*

In the following two sections we will discuss consequences of the hardness results for $\mathrm{DDA}_p^{\rho_N,\rho_\varepsilon}$. More applications lie in deriving **NP**-hardness of response time computation of implicit-deadline tasks as well as **coNP**-hardness of EDF-schedulability of constrained-deadline tasks. But for these problems we will dedicate the whole Chapter 8.

## 7.3 Mixing set

In recent integer programming approaches for production planning the study of simple integer programs, which are part of more sophisticated models has become very successful in practice, see, e.g. Pochet & Wolsey [PW06]. One of these simple integer programs is the so-called *mixing set* [GP01, CDSW07]. The constraint system of a mixing set problem is of the form

$$\begin{array}{rcll} s + a_i y_i & \geq & b_i & \forall i = 1, \dots, n \\ s & \geq & 0 & \\ y_i & \in & \mathbb{Z} & \forall i = 1, \dots, n \\ s & \in & \mathbb{R}. & \end{array} \tag{7.5}$$

where $a_i, b_i \in \mathbb{Q}$. Optimizing a linear function over this mixed integer set can be done in polynomial time, if all $a_i$ are equal to one [GP01, MW03] or if $a_{i+1}/a_i$ is an integer for each $i = 1, \dots, n-1$ [ZdF08], see also [CSW08, CZ08] for subsequent simpler approaches.

Conforti et al. [CSW08] posed the problem, whether one can optimize a linear function over the set of mixed-integer vectors defined by (7.5) also in the general case, to which they refer as the case with *arbitrary capacities*, in polynomial time. In this section, we apply our results on directed Diophantine approximation to show that this problem is **NP**-hard.

For notational reasons it will be more suitable to consider a variant of directed Diophantine approximation as starting point of the reduction, in which $\{\{Q\alpha\}\}^{\uparrow}$ is replaced by $\{\{Q\alpha\}\}^{\downarrow}$. Since $\{\{x\}\}^{\uparrow} = \{\{-x\}\}^{\downarrow}$, all previously derived hardness results for $\mathrm{DDA}_p^{\rho_N,\rho_\varepsilon}$ hold also for this problem, which we term $\mathrm{DDA}_{p,\downarrow}^{\rho_N,\rho_\varepsilon}$.

Directed Diophantine Approximation w.r.t. $\|\cdot\|_p$-norm and rounding down ($\mathrm{DDA}_{p,\downarrow}^{\rho_N,\rho_\varepsilon}$)
Given: $\alpha_1, \dots, \alpha_n \in \mathbb{Q},\ N \in \mathbb{N},\ 0 < \varepsilon \leq 3/\rho_\varepsilon^2$
Distinguish:

- Yes : $\exists Q \in \{\lceil N/2 \rceil, \dots, N\} : \left(\sum_{i=1}^n |Q\alpha_i - \lfloor Q\alpha_i \rfloor|^p\right)^{1/p} \leq \varepsilon$
- No : $\nexists Q \in \{1, \dots, \rho_N \cdot N\} : \left(\sum_{i=1}^n |Q\alpha_i - \lfloor Q\alpha_i \rfloor|^p\right)^{1/p} \leq \rho_\varepsilon \cdot \varepsilon$

Our reduction will follow a Lagrangian relaxation approach, which is common in approximation algorithms, see e.g. [RG96], in order to show a hardness result.

**Theorem 7.13.** *Optimizing a linear function over a mixing set is (weakly)* **NP**-*hard. In other words, given $a, c \in \mathbb{Q}^n, c_s \in \mathbb{Q}, b \in \{0,1\}^n$ and $\delta \in \mathbb{Q}$, it is* **NP**-*hard to decide whether*

$$\min\{c_s s + c^T y \mid s + a_i y \geq b_i\ \forall i = 1, \dots, n; (s, y) \in \mathbb{R}_{\geq 0} \times \mathbb{Z}^n\} \leq \delta.$$

*Proof.* A linear function consists of a *sum* of coefficients, thus we choose $\mathrm{DDA}_{1,\downarrow}^{\rho_N,\rho_\varepsilon}$ as a starting point for the reduction, where a choice of $\rho_N = \rho_\varepsilon > 3$ suffices. Let $(\alpha_1, \dots, \alpha_n; N, \varepsilon)$ be an instance of this problem. Since $\{\{Q\alpha_i\}\}^{\downarrow} = \{\{Q\alpha_i + z\}\}^{\downarrow}$ for $z \in \mathbb{Z}$, we may assume that $\alpha_i > 0$ for all $i = 1, \dots, n$. One can define an integer program for finding a good denominator $Q \in \{1, \dots, N\}$ in a straightforward way.

$$\begin{array}{rcll} \multicolumn{4}{l}{\min \sum_{i=1}^n (Q\alpha_i - y_i)} \\ Q\alpha_i - y_i & \geq & 0 & \forall i = 1, \dots, n \\ Q & \geq & 1 & \\ Q & \leq & N & \\ Q, y_1, \dots, y_n & \in & \mathbb{Z}. & \end{array}$$

The goal is to transform this integer program into a linear optimization problem over a mixing set. So far the constraints $1 \leq Q \leq N$ and $Q \in \mathbb{Z}$ are not admitted. Thus consider the following mixing set.

$$\begin{array}{rcll} Q - 1/\alpha_i \cdot y_i & \geq & 0 & \forall i = 1, \dots, n \\ Q + 0 \cdot y_0 & \geq & 1 & \\ Q - y_{n+1} & \geq & 0 & \\ Q & \in & \mathbb{R}_{\geq 0} & \\ y_0, y_1, \dots, y_n, y_{n+1} & \in & \mathbb{Z}. & \end{array} \tag{7.6}$$

Suppose that the linear optimization problem can be solved in polynomial time. Then we can also solve the linear optimization problem over the non-empty face of the convex hull of the solutions which is induced by the inequality $Q - y_{n+1} \geq 0$, see, e.g., [GLS93]. This enforces $Q$ to be an integer. Furthermore we achieved $Q \geq 1$. It remains to model the constraint $Q \leq N$. We will move this constraint to the objective function, as it is common e.g. in Lagrangian relaxation. Let $OPT$ be the optimum value of the following integer program

$$\min \sum_{i=1}^{n} (Q\alpha_i - y_i) + \frac{\varepsilon}{N} \cdot (Q - N) \tag{$IP$}$$

$$\begin{array}{rl} Q\alpha_i - y_i \geq 0 & \forall i = 1, \dots, n \\ Q \geq 1 & \\ Q, y_i \in \mathbb{Z} & \forall i = 1, \dots, n \end{array}$$

We already discussed that the question whether one has $OPT \leq \varepsilon$ can be reduced to the decision problem, whether a linear function over a mixing set has a solution of at most a value, say $\delta$. Hence it suffices to prove that we are in the YES case of $\mathrm{DDA}_{1,\downarrow}^{\rho_N,\rho_\varepsilon}$ if and only if $OPT \leq \varepsilon$ (for $\rho_N, \rho_\varepsilon \geq 3$). This is done by the following implications

(1) $\exists Q \in \{1, \dots, N\} : \sum_{i=1}^{n} \{\{Q\alpha_i\}\}^{\downarrow} \leq \varepsilon \Rightarrow OPT \leq \varepsilon$

(2) $OPT \leq \varepsilon \Rightarrow \exists Q \in \{1, \dots, 2N\} : \sum_{i=1}^{n} \{\{Q\alpha_i\}\}^{\downarrow} \leq 2\varepsilon$

Note that the conclusion of (2) certifies that we are not in the NO case of $\mathrm{DDA}_{1,\downarrow}^{\rho_N,\rho_\varepsilon}$ and hence, by a lack of choice, we have to be in the YES case, though the obtained $Q$ does not respect the bounds of the YES case.

For (1) let a $Q \in \{1, \dots, N\}$ with $\sum_{i=1}^{n} \{\{Q\alpha_i\}\}^{\downarrow} \leq \varepsilon$ be given. Observe that $(Q, y) = (Q, \lfloor Q\alpha_1 \rfloor, \dots, \lfloor Q\alpha_n \rfloor)$ is an $(IP)$ solution of value

$$\underbrace{\sum_{i=1}^{n} (Q\alpha_i - \lfloor Q\alpha_i \rfloor)}_{\leq \varepsilon} + \frac{\varepsilon}{N} \cdot \underbrace{(Q - N)}_{\leq 0} \leq \varepsilon.$$

For (2) let $(Q, y)$ be an $(IP)$ solution of value at most $\varepsilon$. Since the contribution of $Q\alpha_i - y_i$ to the objective function is non-negative, we must have $\frac{\varepsilon}{N}(Q - N) \leq \varepsilon$ and consequently $Q \leq 2N$. Next observe that

$$\sum_{i=1}^{n} (Q\alpha_i - \lfloor Q\alpha_i \rfloor) \leq \varepsilon - \frac{\varepsilon}{N} \underbrace{(Q - N)}_{\geq -N} \leq 2\varepsilon$$

and the theorem is proven. □

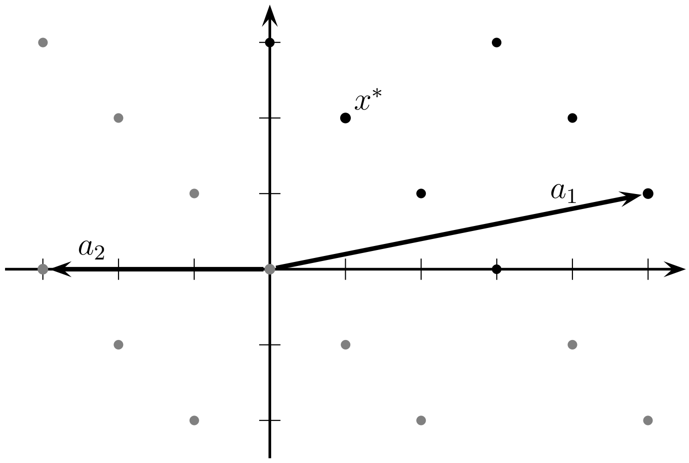


Figure 7.2: Lattice spanned by vectors $a_1 = (5, 1)$ and $a_2 = (-3, 0)$. Black points denote feasible solutions for $\text{SVP}_p^+$, $x^*$ is an optimum solution for $\text{SVP}_2^+$.

## 7.4 The Shortest Vector Problem in the positive orthant

A *lattice* is a set of the form $\Lambda(A) = \{Az \mid z \in \mathbb{Z}^n\}$, where $A \in \mathbb{Q}^{n\times n}$ is a non-singular matrix. The classical *shortest vector problem* (SVP) is to find a nonzero vector in $\Lambda(A)$ with smallest norm. We denote the shortest vector problem w.r.t. $\|\cdot\|_p$-norm by $\text{SVP}_p$. In this section we consider a variant of the shortest vector problem, where the objective is to find a shortest nonzero and non-negative vector. More precisely, the *shortest positive vector problem* ($\text{SVP}_p^+$) is as follows.

Shortest positive Vector Problem ($\text{SVP}_p^+$)
Given: Non-singular matrix $A \in \mathbb{Q}^{n\times n}$
Find:

$$OPT_{\text{SVP}_p^+} = \min\{\|x\|_p \mid x \in (\Lambda(A) \cap \mathbb{R}^n_{\geq 0}), x \neq \mathbf{0}\}$$

See Figure 7.2 for a visualisation. The complexity of the classical shortest vector problem has received a lot of attention by various researchers. Van Emde Boas [vEBil] showed that $\text{SVP}_\infty$ is **NP**-complete. The question whether $\text{SVP}_2$ is hard was open for a long time until, using a randomized reduction Ajtai [Ajt98] succeeded in establishing hardness also for $\text{SVP}_2$. In a series of papers the inapproximability factors for $\text{SVP}_2$ (under standard complexity assumptions) have been improved to: $1 + \frac{1}{n^\varepsilon}$ by Cai & Nerurkar [CN99], $\sqrt{2}-\varepsilon$ due to Micciancio [Mic01], $2^{(\log n)^{1/2-\varepsilon}}$ by Khot [Kho05] and finally $2^{(\log n)^{1-\varepsilon}}$ by Haviv & Regev [HR07]. By randomly embedding the $\|\cdot\|_2$-norm in a higher dimensional space essentially all hardness results can be converted to any $\|\cdot\|_p$-norm [RR06] with $1 \leq p \leq \infty$. However, for $\text{SVP}_\infty$ one can even show $n^{\mathcal{O}(1/\log\log n)}$-hardness [Din02]. This leaves a gap between inapproximability results for $\text{SVP}_\infty$ and $\text{SVP}_p$ for $1 \leq p < \infty$.

Applications of $\text{SVP}_2$ and the LLL algorithm lie for example in cryptography [LO85, Reg04] and integer programming in fixed-dimension [Len83]. The LLL algorithm [LLL82] approximates $\text{SVP}_2$ within a factor of $2^{n/2}$. This factor was improved to $2^{\mathcal{O}(n(\log\log n)^2/\log n)}$ by Schnorr [Sch87] and later to $2^{\mathcal{O}(n\log\log n/\log n)}$ by Ajtai, Kumar & Sivakumar [AKS01].

To the best of our knowledge, $\textsc{Svp}_p^+$ has not been studied before in the literature. By a reduction from $\mathrm{DDA}_\infty^{\rho_N,\rho_\varepsilon}$ we now infer inapproximability for $\textsc{Svp}_p^+$ in any $\|\cdot\|_p$-norm.

**Theorem 7.14.** *There is a universal constant $c' > 0$, such that for all $1 \le p \le \infty$, given a lattice $\Lambda$ and a value $\varepsilon' > 0$ as input, it is* $\mathbf{NP}$*-hard to distinguish between $OPT_{\textsc{Svp}_p^+} \le \varepsilon'$ and $OPT_{\textsc{Svp}_p^+} \ge n^{c'/\log\log n}\varepsilon'$.*

*Proof.* We know from Corollary 7.12 that there is a constant $c > 0$ such that $\mathrm{DDA}_\infty^{\rho_N,\rho_\varepsilon}$ is $\mathbf{NP}$-hard for $\rho_N = \lfloor n^{c/\log\log n}\rfloor$ and $\rho_\varepsilon = 2^n$. Let $(\alpha_1,\dots,\alpha_n;N;\varepsilon)$ be a given $\mathrm{DDA}_\infty^{\rho_N,\rho_\varepsilon}$ instance with $\varepsilon \le 3/\rho_\varepsilon^2$.

We define our lattice by a matrix, which is similar to that one, used to approximate simultaneous Diophantine approximation with the help of the LLL algorithm for the classical *Shortest Vector Problem* [LLL82].

$$A = \begin{pmatrix} 1/n & 0 & \dots & 0 & 0 & -\alpha_1/n \\ 0 & 1/n & \dots & 0 & 0 & -\alpha_2/n \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 1/n & 0 & -\alpha_{n-1}/n \\ 0 & 0 & \dots & 0 & 1/n & -\alpha_n/n \\ 0 & 0 & \dots & 0 & 0 & \varepsilon/N \end{pmatrix}$$

Let $\Lambda$ be the lattice, generated by the columns of $A$. It remains to show the upcoming two implications; the theorem then follows from $\varepsilon' := 2\varepsilon$ and $c' := c/2$.

- $\textsc{Yes}: \exists Q \in \{\lceil N/2\rceil,\dots,N\} : \{\{Q\alpha\}\}^\uparrow \le \varepsilon \Rightarrow OPT_{\textsc{Svp}_p^+} \le 2\varepsilon$
- $\textsc{No}: \nexists Q \in \{1,\dots,\lfloor n^{c/\log\log n}\rfloor \cdot N\} : \{\{Q\alpha\}\}^\uparrow \le 2^n\varepsilon \Rightarrow OPT_{\textsc{Svp}_p^+} > \lfloor n^{c/\log\log n}\rfloor \cdot \varepsilon$

For the first implication let $Q \in \{\lceil N/2\rceil,\dots,N\}$ with $\max_{i=1,\dots,n}(\lceil Q\alpha_i\rceil - Q\alpha_i) \le \varepsilon$ be a $\mathrm{DDA}_\infty^{\rho_N,\rho_\varepsilon}$ solution. Then

$$x := A\cdot(\lceil Q\alpha_1\rceil,\dots,\lceil Q\alpha_n\rceil,Q)^T = \left(\frac{1}{n}(\lceil Q\alpha_1\rceil - Q\alpha_1),\dots,\frac{1}{n}(\lceil Q\alpha_n\rceil - Q\alpha_n),\varepsilon\frac{Q}{N}\right)^T$$

is a positive vector in the lattice of length

$$\|x\|_p \le \frac{1}{n}\sum_{i=1}^{n}\underbrace{(\lceil Q\alpha_i\rceil - Q\alpha_i)}_{\le\varepsilon} + \varepsilon\cdot\underbrace{\frac{Q}{N}}_{\le 1} \le 2\varepsilon.$$

Vice versa, let $x \in \Lambda$ be a non-negative, non-zero vector in the lattice with $\|x\|_p \le \lfloor n^{c/\log\log n}\rfloor\varepsilon$, which is generated by $(z_1,\dots,z_n,Q) \in \mathbb{Z}^n$. W.l.o.g. we assume $Q \ge 0$, otherwise replace $Q$ by $-Q$. If $Q = 0$, then $\|x\|_p \ge 1/n \gg \lfloor n^{c/\log\log n}\rfloor \cdot \varepsilon$ for $n$ large enough since $\varepsilon \le 3/\rho_\varepsilon^2 \le 3/2^{2n}$. Thus $Q \ge 1$. On the other hand we have $Q\cdot\frac{\varepsilon}{N} \le \|x\|_p \le \lfloor n^{c/\log\log n}\rfloor\varepsilon$, hence $Q \le \lfloor n^{c/\log\log n}\rfloor \cdot N$. It remains to bound the approximation error. But we have $\frac{1}{n}(z_i - Q\alpha_i) \le \lfloor n^{c/\log\log n}\rfloor\varepsilon$ and consequently $\{\{Q\alpha_i\}\}^\uparrow \le n^{1+c/\log\log n}\cdot\varepsilon \le 2^n\cdot\varepsilon$. The claim then follows. □

We conclude

**Corollary 7.15.** *There exists a constant $c > 0$, s.t. for all $\|\cdot\|_p$-norms ($1 \le p \le \infty$) it is* $\mathbf{NP}$*-hard to approximate $\textsc{Svp}_p^+$ within a factor of $n^{c/\log\log n}$.*

# Chapter 8

# Hardness of Schedulability

After the excursion to the geometry of numbers in the last chapter, we are now back to the field of periodic scheduling. In this chapter we investigate the complexity of testing the feasibility of implicit-deadline tasks under RM-scheduling and of explicit-deadline tasks under EDF-scheduling.

We are given constrained-deadline tasks $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$, each $\tau \in \mathcal{S}$ with running time $c(\tau)$, period $p(\tau)$ and deadline $d(\tau) \leq p(\tau)$. For EDF-schedulability of these tasks one has to verify whether

$$\forall t \geq 0 : \sum_{\tau \in \mathcal{S}} \left( \left\lfloor \frac{t - d(\tau)}{p(\tau)} \right\rfloor + 1 \right) \cdot c(\tau) \leq t, \tag{8.1}$$

see [BMR90]. If the deadlines are *implicit*, i.e. $d(\tau) = p(\tau)$ for all $\tau \in \mathcal{S}$, then the response time for $\tau_j$ in a Rate-monotonic schedule is the smallest $r \geq 0$ such that

$$c(\tau_j) + \sum_{i=1}^{j-1} \left\lceil \frac{r}{p(\tau_i)} \right\rceil c(\tau_i) \leq r. \tag{8.2}$$

holds, given that the tasks are ordered according to their priorities, i.e. $p(\tau_1) \leq \ldots \leq p(\tau_n)$. Here we will prove that both conditions are hard to evaluate unless $\mathbf{NP} = \mathbf{P}$[1]. We know from the last chapter that testing conditions that involve directed rounding is **NP**-hard. However, a closer look reveals that $t$ and $r$ in (8.1) and (8.2) are both ranging over the *rational* numbers, not over the integers as it was the case for directed Diophantine approximation. The first claim that we have to show is that directed Diophantine approximation remains intractable, even if we allow the denominator $Q$ to be an arbitrary rational number. We will now see that this is true — at least if we introduce weights...

## 8.1 Weighted directed Diophantine approximation

We begin by formally introducing a version of directed Diophantine approximation, where one asks for a possibly fractional denominator $Q$. For the sake of simplicity, we only introduce this problem w.r.t. the $\|\cdot\|_1$-norm.

[1] A preliminary version of the hardness result for (8.2) appeared already in a slightly weaker form in [ER08b].

Weighted Directed Diophantine Approximation w.r.t. $\|\cdot\|_1$-norm ($\mathrm{DDA}_{1,w}^{\rho_N,\rho_\varepsilon}$)
Given: $\alpha_1,\ldots,\alpha_n \in \mathbb{Q} \cap [1,2]$, $w_1,\ldots,w_n \in \mathbb{Q}_+$, $N \in \mathbb{N}$
Distinguish:

- Yes : $\exists Q \in [\lceil N/2\rceil, N] : \sum_{i=1}^n w_i(\lceil Q\alpha_i\rceil - Q\alpha_i) \leq \varepsilon$
- No : $\nexists Q \in [1, \rho_N \cdot N] : \sum_{i=1}^n w_i(\lceil Q\alpha_i\rceil - Q\alpha_i) \leq \rho_\varepsilon \cdot \varepsilon$

While in the previous chapter the input numbers were arbitrary rationals, from now on we demand the $\alpha_i$'s to be in the interval $[1,2]$. Since $\{\{Q\alpha_i\}\}^\uparrow = \{\{Q(\alpha_i+z)\}\}^\uparrow$ for $Q, z \in \mathbb{Z}$, we may assume w.l.o.g. that $1 \leq \alpha_i \leq 2$ for any $\mathrm{DDA}_1^{\rho_N,\rho_\varepsilon}$ instance $(\alpha_1,\ldots,\alpha_n;\varepsilon;N)$. Recall Corollary 7.12 that it is **NP**-hard to distinguish between the two cases

- Yes: $\exists Q \in \{\lceil N/2\rceil,\ldots,N\} : \sum_{i=1}^n(\lceil Q\alpha_i\rceil - Q\alpha_i) \leq \varepsilon$
- No: $\nexists Q \in \{1,\ldots,\lfloor n^{c_2/\log\log n}\rfloor \cdot N\} : \sum_{i=1}^n(\lceil Q\alpha_i\rceil - Q\alpha_i) \leq 2^{n^{c_1}} \cdot \varepsilon$

where $c_1, c_2$ are constants.

We now convey this hardness result to the weighted (but fractional) case of directed Diophantine approximation. Note that still $\rho_N, \rho_\varepsilon \in \mathbb{N}$.

**Theorem 8.1** (Gap preserving reduction from $\mathrm{DDA}_1^{\rho_N,\rho_\varepsilon}$ to $\mathrm{DDA}_{1,w}^{\rho_N,\rho_\varepsilon}$)**.** *There is a polynomial time transformation, taking a* $\mathrm{DDA}_1^{\rho_N,\rho_\varepsilon}$ *instance* $(\alpha_1,\ldots,\alpha_n;\varepsilon;N)$ *with* $1 \leq \alpha_i \leq 2$ $\forall i = 1,\ldots,n$ *as input and yielding a* $\mathrm{DDA}_{1,w}^{\rho_N,\rho_\varepsilon}$ *instance* $(\alpha_0,\ldots,\alpha_n;w_0,\ldots,w_n;\varepsilon;N)$ *such that*

- Yes*:* $\exists Q \in \{\lceil N/2\rceil,\ldots,N\} : \sum_{i=1}^n(\lceil Q\alpha_i\rceil - Q\alpha_i) \leq \varepsilon$
  $\Rightarrow \exists Q \in [\lceil N/2\rceil, N] : \sum_{i=0}^n w_i(\lceil Q\alpha_i\rceil - Q\alpha_i) \leq \varepsilon$
- No*:* $\nexists Q \in \{1,\ldots,\rho_N \cdot N\} : \sum_{i=1}^n(\lceil Q\alpha_i\rceil - Q\alpha_i) \leq \rho_\varepsilon \cdot \varepsilon$
  $\Rightarrow \nexists Q \in [1, \rho_N \cdot N] : \sum_{i=0}^n w_i(\lceil Q\alpha_i\rceil - Q\alpha_i) \leq \rho_\varepsilon \cdot \varepsilon$

*Proof.* Let $(\alpha_1,\ldots,\alpha_n;N;\varepsilon)$ be the given $\mathrm{DDA}_1^{\rho_N,\rho_\varepsilon}$ instance. Since the $\alpha_i$'s are rational numbers, we can write them as $\alpha_i = \frac{a_i}{b_i}$ with pairwise co-prime integers $a_i, b_i \in \mathbb{N}$. Our $\mathrm{DDA}_{1,w}^{\rho_N,\rho_\varepsilon}$ instance consists of the same bounds $\varepsilon$ and $N$; the numbers $\alpha_1,\ldots,\alpha_n$ are equipped with unit weights $w_1 = \ldots = w_n = 1$. But furthermore we add an extra number $\alpha_0 := 1$ with a very high weight of $w_0 := 2 \cdot \max\{a_i \mid i = 1,\ldots,n\} \cdot \varepsilon \cdot \rho_\varepsilon$. Intuitively the weight $w_0$ is large enough, such that any reasonable $\mathrm{DDA}_{1,w}^{\rho_N,\rho_\varepsilon}$ solution $Q$ to this instance must be an exact multiple of $\alpha_0 = 1$, thus it must be integer.

Yes-case: Clearly Yes instances for $\mathrm{DDA}_1^{\rho_N,\rho_\varepsilon}$ are mapped to Yes instances of $\mathrm{DDA}_{1,w}^{\rho_N,\rho_\varepsilon}$ by simply using the same solution $Q$. This is the case since given a $Q \in \{\lceil N/2\rceil,\ldots,N\}$ that matches the conditions of the Yes case for $\mathrm{DDA}_1^{\rho_N,\rho_\varepsilon}$, one has

$$\sum_{i=0}^n w_i(\lceil Q\alpha_i\rceil - Q\alpha_i) = w_0 \cdot \underbrace{(\lceil Q\rceil - Q)}_{=0} + \sum_{i=1}^n 1 \cdot (\lceil Q\alpha_i\rceil - Q\alpha_i) \leq \varepsilon.$$

No-case: Now suppose that we have a $Q \in [1, \rho_N \cdot N]$ with $\sum_{i=0}^n w_i(\lceil Q\alpha_i\rceil - Q\alpha_i) \leq \rho_\varepsilon \cdot \varepsilon$. Increase $Q$ until $Q\alpha_j \in \mathbb{Z}$ for at least one $j \in \{0,\ldots,n\}$, while the approximation error can only decrease (or leave $Q$ unchanged if this is already the case). If $Q$ is then an integer, we are done. Otherwise, we may write $Q\alpha_j = Q\frac{a_j}{b_j} =: z \in \mathbb{Z}$, thus $Q = \frac{zb_j}{a_j} \in \mathbb{Z}\frac{1}{a_j}$. We obtain

$$\sum_{i=0}^{n} w_i(\lceil Q\alpha_i \rceil - Q\alpha_i) \geq w_0 \cdot (\lceil Q \rceil - Q) \geq w_0 \cdot \frac{1}{a_j} > \rho_\varepsilon \cdot \varepsilon$$

by the choice of $w_0$. The contradiction yields that $Q \in \mathbb{N}$ and the claim follows. □

We conclude

**Corollary 8.2** (Hardness of $\mathrm{DDA}_{1,w}^{\rho_N,\rho_\varepsilon}$)**.** *There exists a constant $c_2 > 0$, such that for all $c_1 > 0$ the problem $\mathrm{DDA}_{1,w}^{\rho_N,\rho_\varepsilon}$ is* **NP***-hard for $\rho_N = n^{c_2/\log\log n}$ and $\rho_\varepsilon = 2^{n^{c_1}}$. In other words, given an instance $(\alpha_1, \ldots, \alpha_n; w_1, \ldots, w_n; \varepsilon; N)$ with $1 \leq \alpha_i \leq 2\ \forall i = 1, \ldots, n$ it is* **NP***-hard to distinguish*

- Yes*: $\exists Q \in [\lceil N/2 \rceil, N] : \sum_{i=1}^{n} w_i(\lceil Q\alpha_i \rceil - Q\alpha_i) \leq \varepsilon$*
- No*: $\nexists Q \in [1, n^{c_2/\log\log n} \cdot N] : \sum_{i=1}^{n} w_i(\lceil Q\alpha_i \rceil - Q\alpha_i) \leq 2^{n^{c_1}} \cdot \varepsilon$*

A weaker version for intractability of weighted Diophantine approximation can also be found in [ER08b]. The weights used in the proof are astronomically large. We conjecture that this is not needed.

**Conjecture 8.3.** *There are $\rho_N, \rho_\varepsilon > 1$ such that $\mathrm{DDA}_{1,w}^{\rho_N,\rho_\varepsilon}$ is* **NP***-hard for unit weights.*

For the sake of completeness, we want to mention that the variant, where the approximation error is measured as the distance to the next *smaller* integer, is hard as well. Define

Weighted Directed Diophantine Approximation w.r.t. $\|\cdot\|_1$-norm, with rounding down ($\mathrm{DDA}_{1,w,\downarrow}^{\rho_N,\rho_\varepsilon}$)
Given: $\alpha_1, \ldots, \alpha_n \in \mathbb{Q} \cap [1, 2]$, $w_1, \ldots, w_n \in \mathbb{Q}_+$, $N \in \mathbb{N}$
Distinguish:

- Yes : $\exists Q \in [\lceil N/2 \rceil, N] : \sum_{i=1}^{n} w_i(Q\alpha_i - \lfloor Q\alpha_i \rfloor) \leq \varepsilon$
- No : $\nexists Q \in [1, \rho_N \cdot N] : \sum_{i=1}^{n} w_i(Q\alpha_i - \lfloor Q\alpha_i \rfloor) \leq \rho_\varepsilon \cdot \varepsilon$

By essentially reproducing the proof of Theorem 8.1, one can obtain a gap preserving reduction from $\mathrm{DDA}_{1,\downarrow}^{\rho_N,\rho_\varepsilon}$ to $\mathrm{DDA}_{1,w,\downarrow}^{\rho_N,\rho_\varepsilon}$, hence

**Corollary 8.4.** *There exists a constant $c_2 > 0$, such that for all $c_1 > 0$ the problem $\mathrm{DDA}_{1,w,\downarrow}^{\rho_N,\rho_\varepsilon}$ is* **NP***-hard with $\rho_N = n^{c_2/\log\log n}$ and $\rho_\varepsilon = 2^{n^{c_1}}$.*

## 8.2 Hardness of EDF-schedulability

In the following, let $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$ be a constrained-deadline task system, i.e. each task $\tau_i = (c(\tau_i), d(\tau_i), p(\tau_i))$ releases a job of running time $c(\tau_i)$ at $z \cdot p(\tau_i)$ with absolute

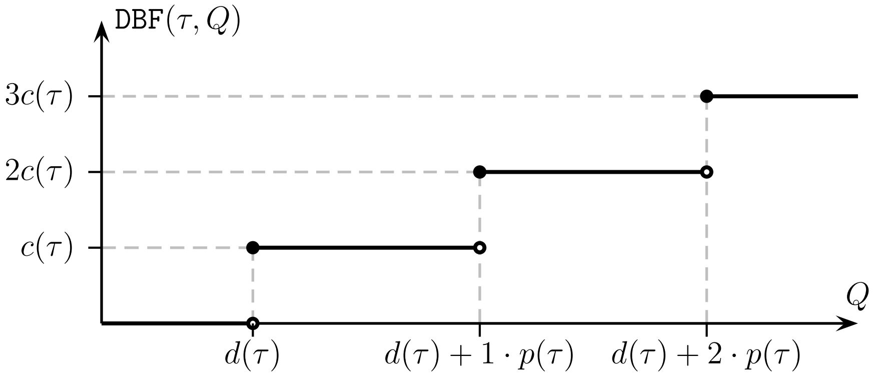


Figure 8.1: Visualization of $\texttt{DBF}(\tau, Q)$

deadline $z \cdot p(\tau_i) + d(\tau_i)$ for each $z \in \mathbb{N}_0$. Here we consider EDF-scheduling, which is an optimal dynamic-priority policy in this setting. Define

$$\texttt{DBF}(\tau, Q) = \left( \left\lfloor \frac{Q - d(\tau)}{p(\tau)} \right\rfloor + 1 \right) \cdot c(\tau)$$

as the *demand bound function* of a constrained-deadline task $\tau$ at time $Q \geq 0$. Note that

$$\left\lfloor \frac{Q - d(\tau)}{p(\tau)} \right\rfloor + 1$$

gives the number of jobs of $\tau$ that have both, their release time and deadline in the interval $[0, Q]$, see Figure 8.1. More general

$$\texttt{DBF}(\mathcal{S}, Q) = \sum_{\tau \in \mathcal{S}} \texttt{DBF}(\tau, Q)$$

gives the demand of the whole task system $\mathcal{S}$. Obviously the EDF-schedule cannot be feasible on a unit-speed processor, if there is a $Q \geq 0$ with $\texttt{DBF}(\mathcal{S}, Q) > Q$. In Baruah, Mok & Rosier [BMR90] it was shown that this necessary condition is also sufficient. Hence $\mathcal{S}$ is EDF-schedulable on a unit-speed processor if and only if

$$\forall Q \geq 0 : \texttt{DBF}(\mathcal{S}, Q) \leq Q.$$

See Figure 8.2 for an illustration.

Much effort has been spent on developing sufficient polynomial or exact pseudo-polynomial time tests for the above condition, see [AS05, BCGM99, BMR90, CKT02, Dev03]. Albers & Slomka [AS04] e.g. gave an FPTAS for approximating the speed of a processor, needed to make the EDF-schedule of $\mathcal{S}$ feasible. Similar to the result of [FB05] for RM-scheduling this is obtained by ignoring the rounding operation in the function $\texttt{DBF}(\tau, Q)$ as soon as the term $\frac{Q-d(\tau)}{p(\tau)}$ exceeds $1/\varepsilon$. This yields a function $\texttt{DBF}'(\tau, Q)$ with $\texttt{DBF}(\tau, Q) \leq \texttt{DBF}'(\tau, Q) \leq (1+\varepsilon) \cdot \texttt{DBF}(\tau, Q)$. For the obtained accumulated demand bound function $\texttt{DBF}'(\mathcal{S}, Q)$, the value of only $\mathcal{O}(n/\varepsilon)$ many control points need to be evaluated, which can be done in time $\mathcal{O}(n(\log n)/\varepsilon)$.

But none of the algorithms suggested in these papers was able decide EDF-schedulability on a unit speed processor correctly and in polynomial time for all instances. We

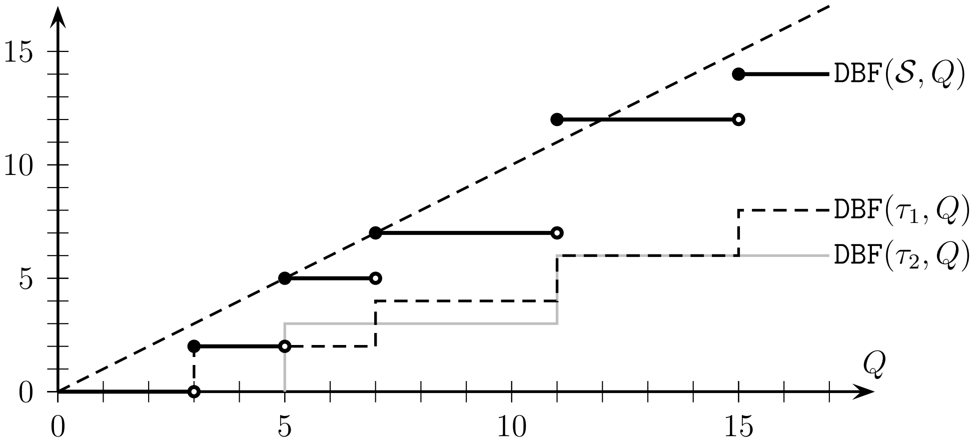


Figure 8.2: Constrained-deadline task system $\mathcal{S} = \{\tau_1, \tau_2\}$ with $\tau_1 = (2, 3, 4)$, $\tau_2 = (3, 5, 6)$, using notation $\tau = (c(\tau), d(\tau), p(\tau))$. One has $\texttt{DBF}(\mathcal{S}, Q) > Q$ for $Q = 11$, thus $\mathcal{S}$ is not EDF-schedulable.

now give the underlying theoretical reason for this, by proving its intractability. This was posed as Problem 2 in the list of *Open Problems in Real-time Scheduling* by Baruah & Pruhs [BP09]. We show **coNP**-hardness of testing EDF-schedulability by a reduction from the **NP**-hard problem $\text{DDA}_{1,w,\downarrow}^{\rho_N,\rho_\varepsilon}$ (gap parameters $\rho_N = \rho_\varepsilon = 4$ fully suffice). Here YES cases are mapped to NO cases and vice versa.

Intuitively this is done as follows: Let $(\alpha_1, \ldots, \alpha_n; w_1, \ldots, w_n; N; \varepsilon)$ be a $\text{DDA}_{1,w,\downarrow}^{\rho_N,\rho_\varepsilon}$ instance. The first idea is to create implicit-deadline tasks $\tau_1, \ldots, \tau_n$ with $p(\tau_i) = d(\tau_i) = \frac{1}{\alpha_i}$. Then we have

$$\left\lfloor \frac{Q - d(\tau_i)}{p(\tau_i)} \right\rfloor + 1 = \lfloor Q\alpha_i \rfloor$$

hence a $Q$ that maximizes $\texttt{DBF}(\mathcal{S}, Q)/Q$, minimizes the approximation error and gives a good denominator. But this $Q$ might be arbitrarily large, in fact it would be a common multiple of all $p(\tau_i)$'s. To enforce an EDF-schedulability oracle to find a small $Q$, we add one special task $\tau_0$ which has a deadline of $N/2$ and a sufficiently large period (we may imagine $p(\tau_0) = \infty$). Then the quantity $\texttt{DBF}(\tau_0, Q)/Q$ contributes significantly to $\texttt{DBF}(\mathcal{S}, Q)/Q$ only if $Q$ is of order $N$. See also Figure 8.3 for an illustration.

**Theorem 8.5.** *Given an instance consisting of rational numbers $\alpha_1, \ldots, \alpha_n \in \mathbb{Q}_+$, weights $w_1, \ldots, w_n \in \mathbb{Q}_+$, a bound $N \in \mathbb{N}_{\geq 2}$ and an error bound $\varepsilon > 0$, we can find a constrained-deadline task system $\mathcal{S}$ consisting of $n + 1$ tasks in polynomial time such that*

- YES*: $\exists Q \in [\lceil N/2 \rceil, N] : \sum_{i=1}^n w_i(Q\alpha_i - \lfloor Q\alpha_i \rfloor) \leq \varepsilon \Rightarrow \mathcal{S}$ not EDF-schedulable*
- NO*: $\nexists Q \in [\lceil N/2 \rceil, 3N] : \sum_{i=1}^n w_i(Q\alpha_i - \lfloor Q\alpha_i \rfloor) \leq 3\varepsilon \Rightarrow \mathcal{S}$ EDF-schedulable*

*Furthermore $n$ tasks in $\mathcal{S}$ have implicit-deadlines.*

*Proof.* A set of tasks is EDF-schedulable on a processor of speed $\beta > 0$ if and only if the tasks with running times scaled by $\frac{1}{\beta}$ are feasible on a unit speed processor. Thus we may assume to have an oracle for the test

$$\forall Q \geq 0 : \sum_{\tau \in \mathcal{S}} \left( \left\lfloor \frac{Q - d(\tau)}{p(\tau)} \right\rfloor + 1 \right) \cdot c(\tau) \leq \beta \cdot Q$$

Let $N \in \mathbb{N}, \alpha_1, \ldots, \alpha_n, w_1, \ldots, w_n \in \mathbb{Q}_+, \varepsilon > 0$ be the $\mathrm{DDA}_{1,w,\downarrow}$ instance. We choose a constrained-deadline task system $\mathcal{S}$ consisting of $n+1$ tasks

$$\begin{aligned} \tau_i &= (c(\tau_i), d(\tau_i), p(\tau_i)) := \left( w_i, \frac{1}{\alpha_i}, \frac{1}{\alpha_i} \right) \quad \forall i = 1, \ldots, n \\ \tau_0 &= (c(\tau_0), d(\tau_0), p(\tau_0)) := (3\varepsilon, \lceil N/2 \rceil, 12N) \end{aligned}$$

and processor speed

$$\beta := \frac{\varepsilon}{N} + \sum_{i=1}^{n} w_i \alpha_i$$

<u>Yes-case:</u> Suppose that we have a $Q \in [\lceil N/2 \rceil, N]$ with $\sum_{i=1}^{n} w_i(Q\alpha_i - \lfloor Q\alpha_i \rfloor) \leq \varepsilon$. Then

$$\begin{aligned} \mathtt{DBF}(\{\tau_0, \ldots, \tau_n\}, Q) &= \mathtt{DBF}(\tau_0, Q) + \sum_{i=1}^{n} \left( \left\lfloor \frac{Q - d(\tau_i)}{p(\tau_i)} \right\rfloor + 1 \right) c(\tau_i) \\ &= 3\varepsilon + \sum_{i=1}^{n} \lfloor Q\alpha_i \rfloor w_i \\ &\overset{(*)}{\geq} 3\varepsilon + \left( \left( \sum_{i=1}^{n} Q\alpha_i w_i \right) - \varepsilon \right) \\ &= 2\varepsilon + Q \sum_{i=1}^{n} \alpha_i w_i \\ &\overset{(**)}{>} Q \cdot \underbrace{\left( \frac{\varepsilon}{N} + \sum_{i=1}^{n} \alpha_i w_i \right)}_{=\beta} \\ &= \beta Q \end{aligned}$$

Here we use $\sum_{i=1}^{n} w_i(Q\alpha_i - \lfloor Q\alpha_i \rfloor) \leq \varepsilon$ in $(*)$ and $Q \leq N < 2N$ in $(**)$. Thus the task system $\mathcal{S}$ is not EDF-schedulable (on a processor of speed $\beta$).

<u>No-case:</u> Next we assume that $\mathcal{S}$ is not EDF-schedulable (on a processor of speed $\beta$). Then there is a $Q > 0$ such that $\mathtt{DBF}(\{\tau_0, \ldots, \tau_n\}, Q) > \beta Q$. We need to show that $Q \in [\lceil N/2 \rceil, 3N]$ and $\sum_{i=1}^{n} w_i(Q\alpha_i - \lfloor Q\alpha_i \rfloor) \leq 3\varepsilon$.

Observe that using the definition of $\beta$ and $\lfloor Q\alpha_i \rfloor \leq Q\alpha_i$, one has

$$\begin{aligned}
\texttt{DBF}(\tau_0, Q) &= \texttt{DBF}(\mathcal{S}, Q) - \texttt{DBF}(\{\tau_1, \dots, \tau_n\}, Q) \\
&> \beta Q - \sum_{i=1}^{n} \lfloor Q\alpha_i \rfloor w_i \\
&\geq \beta Q - Q \sum_{i=1}^{n} \alpha_i w_i \\
&= \beta Q - Q \underbrace{\left( \frac{\varepsilon}{N} + \sum_{i=1}^{n} \alpha_i w_i \right)}_{=\beta} + Q\frac{\varepsilon}{N} \\
&= Q\frac{\varepsilon}{N}
\end{aligned}$$

Since $\tau_0$ has its first deadline at $d(\tau_0) = \lceil N/2 \rceil$ and $\texttt{DBF}(\tau_0, Q) > 0$ we must have $Q \geq \lceil N/2 \rceil$. Suppose for contradiction that already the second deadline of $\tau_0$ occurred before $Q$, i.e. $Q \geq p(\tau_0) = 12N$. Then

$$\texttt{DBF}(\tau_0, Q) \leq c(\tau_0) \cdot \left\lceil \frac{Q}{p(\tau_0)} \right\rceil \leq 2 \cdot 3\varepsilon \cdot \frac{Q}{12N} < Q\frac{\varepsilon}{N},$$

leading to a contradiction. Hence, till time $Q$ exactly one deadline of $\tau_0$ has passed, thus $\texttt{DBF}(\tau_0, Q) = 3\varepsilon$. But we already inferred the bound $\texttt{DBF}(\tau_0, Q) > Q\frac{\varepsilon}{N}$, thus even $Q < 3N$. Finally

$$\sum_{i=1}^{n} w_i(Q\alpha_i - \lfloor Q\alpha_i \rfloor) = Q \underbrace{\sum_{i=1}^{n} \alpha_i w_i}_{<\beta} - (\texttt{DBF}(\mathcal{S}, Q) - \texttt{DBF}(\tau_0, Q)) \leq \underbrace{Q\beta - \texttt{DBF}(\mathcal{S}, Q)}_{<0} + 3\varepsilon \leq 3\varepsilon$$

and the claim follows. □

Combining Theorem 8.5 and Corollary 8.4 we obtain

**Corollary 8.6.** *Given a set $\mathcal{S} = \{\tau_1, \dots, \tau_n\}$ $(\tau_i = (c(\tau_i), d(\tau_i), p(\tau_i)))$ of constrained-deadline tasks with rational numbers $0 \leq c(\tau_i) \leq d(\tau_i) \leq p(\tau_i)$, it is (weakly)* **coNP***-hard to decide, whether $\mathcal{S}$ is EDF-schedulable, i.e. testing the condition*

$$\forall Q \geq 0 : \sum_{\tau \in \mathcal{S}} \left( \left\lfloor \frac{Q - d(\tau)}{p(\tau)} \right\rfloor + 1 \right) \cdot c(\tau) \leq Q,$$

*is (weakly)* **coNP***-hard. This holds even if for all but one i, one has $d(\tau_i) = p(\tau_i)$.*

## 8.3 Hardness of response time computation

Now we again consider Rate-monotonic scheduling of tasks with implicit-deadlines that means $d(\tau_i) = p(\tau_i)$ for all $\tau_i \in \mathcal{S}$. Recall that for testing feasibility of an RM-schedule, usually the response times are computed and then compared to the periods. Suppose

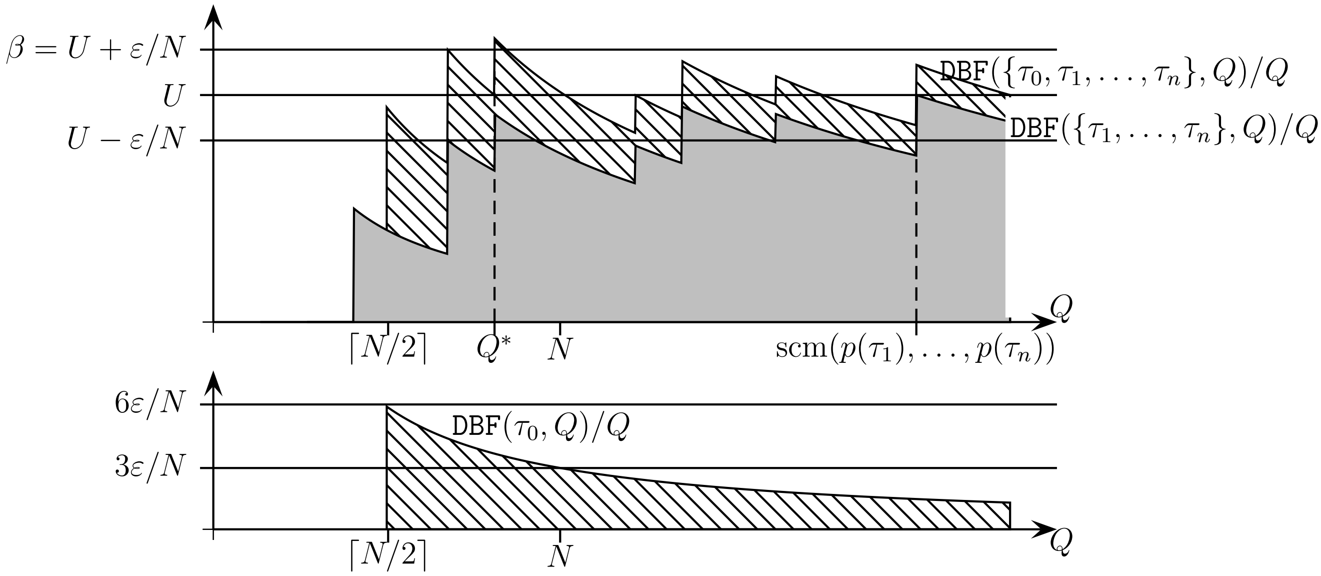


Figure 8.3: In the upper picture the gray shaded function $\mathtt{DBF}(\{\tau_1, \ldots, \tau_n\}, Q)/Q$ is depicted. Intuitively this function is larger, the closer the values $\frac{Q}{p(\tau_i)}$ are to the next smaller integer. It can never exceed $U := u(\{\tau_1, \ldots, \tau_n\})$, but if the approximation error of $Q$ w.r.t. $\alpha_1, \ldots, \alpha_n$ is less than $\varepsilon$, then $\mathtt{DBF}(\{\tau_1, \ldots, \tau_n\}, Q)/Q \geq U - \frac{\varepsilon}{Q}$. In the lower picture we see $\mathtt{DBF}(\tau_0, Q)/Q$, giving the demand from the auxiliary task $\tau_0$. In the interval $\lceil N/2 \rceil \leq Q \leq N$ this value is between $6\varepsilon/N$ and $3\varepsilon/N$. Thus, if additionally $Q = Q^*$ is a good approximation, then the demand of $\tau_0$ pushes $\mathtt{DBF}(\{\tau_0, \ldots, \tau_n\}, Q)/Q$ over the threshold of $\beta = U + \frac{\varepsilon}{N}$. On the other side for $Q > 3N$, the value of $\mathtt{DBF}(\tau_0, Q)/Q$ is to small for this purpose.

that $\tau_{n+1}$ is the task with the highest period (thus it has the lowest priority in an RM-schedule), then the response time $r(\tau_{n+1})$ of this task is the smallest number $r(\tau_{n+1}) \geq 0$ such that

$$c(\tau_{n+1}) + \sum_{i=1}^{n} \left\lceil \frac{r(\tau_{n+1})}{p(\tau_i)} \right\rceil c(\tau_i) \leq r(\tau_{n+1}).$$

Of course any solution to this inequality gives an upper bound on the response time.

For the upcoming inapproximability result of response time computation, working with different gap parameters $\rho_N$ and $\rho_\varepsilon$ would not yield a significantly stronger result. Thus for the sake of a simple representation, we restrict the gap preserving reduction to a common gap parameter $\rho := \rho_\varepsilon = \rho_N$.

**Theorem 8.7.** *There is a polynomial time reduction, taking a* $\mathrm{DDA}_{1,w}^{\rho,\rho}$ *instance* $(\alpha_1, \ldots, \alpha_n; w_1, \ldots, w_n; \varepsilon; N)$ $(1 \leq \alpha_i \leq 2\ \forall i = 1, \ldots, n;\ N \geq 2)$ *and yielding a set of implicit-deadline tasks* $\mathcal{S} = \{\tau_1, \ldots, \tau_{n+1}\}$ *such that*

- Yes*:* $\exists Q \in [\lceil N/2 \rceil, N] : \sum_{i=1}^{n} w_i(Q\alpha_i - \lfloor Q\alpha_i \rfloor) \leq \varepsilon \Rightarrow r(\tau_{n+1}) \leq p(\tau_{n+1})$

- No*:* $\nexists Q \in [1, \rho \cdot N] : \sum_{i=1}^{n} w_i(Q\alpha_i - \lfloor Q\alpha_i \rfloor) \leq 4\rho \cdot \varepsilon \Rightarrow r(\tau_{n+1}) > \rho \cdot p(\tau_{n+1})$

*where* $r(\tau_{n+1})$ *gives the response time of* $\tau_{n+1}$ *in a Rate-monotonic schedule of* $\mathcal{S}$.

*Proof.* Choose

$$\delta := \frac{4\varepsilon}{N} + \sum_{i=1}^{n} w_i \alpha_i.$$

We define an instance of $n+1$ implicit-deadline tasks via

$$\begin{aligned} (c(\tau_i), p(\tau_i)) &:= \left( \frac{w_i}{\delta}, \frac{1}{\alpha_i} \right) \quad \forall i = 1, \ldots, n \\ (c(\tau_{n+1}), p(\tau_{n+1})) &:= \left( \frac{\varepsilon}{\delta}, N \right) \end{aligned}$$

First recall that for $i = 1, \ldots, n$ one has $p(\tau_i) = \frac{1}{\alpha_i} \leq 1 < N = p(\tau_{n+1})$, thus $\tau_{n+1}$ gets assigned the lowest priority in an RM-schedule. The intuition behind the system is as follows: The tasks $\tau_1, \ldots, \tau_n$ are chosen such that $Q$ is a good Diophantine approximation to $\alpha_1, \ldots, \alpha_n$ if and only the function $\sum_{i=1}^{n} \lceil \frac{Q}{p(\tau_i)} \rceil \cdot c(\tau_i)$ is close to $Q$. The contribution of $\tau_{n+1}$ then controls the approximation error.

<u>Yes-case:</u> Suppose that we have a $Q \in [\lceil N/2 \rceil, N]$ with $\sum_{i=1}^{n} w_i(\lceil Q\alpha_i \rceil - Q\alpha_i) \leq \varepsilon$

(used in $(*)$). We show that $r(\tau_{n+1}) \leq Q \leq N$.

$$\begin{aligned}
c(\tau_{n+1}) + \sum_{i=1}^{n} \left\lceil \frac{Q}{p(\tau_i)} \right\rceil \cdot c(\tau_i) &= \frac{\varepsilon}{\delta} + \sum_{i=1}^{n} \lceil Q\alpha_i \rceil \cdot \frac{w_i}{\delta} \\
&\overset{(*)}{\leq} \frac{\varepsilon}{\delta} + \frac{1}{\delta}\left[\sum_{i=1}^{n} Q\alpha_i w_i + \varepsilon\right] \\
&= \frac{1}{\delta}\left[\sum_{i=1}^{n} Q\alpha_i w_i + 2\varepsilon\right] \\
&\overset{(**)}{\leq} \frac{Q}{\delta}\underbrace{\left[\sum_{i=1}^{n} \alpha_i w_i + \frac{4\varepsilon}{N}\right]}_{=\delta} \\
&= Q
\end{aligned}$$

Here $(**)$ follows from $Q \geq N/2$, thus $Q$ is an upper bound on the response time.

<u>No-case:</u> Suppose $Q \leq \rho \cdot p(\tau_{n+1})$ is the response time, hence

$$c(\tau_{n+1}) + \sum_{i=1}^{n} \left\lceil \frac{Q}{p(\tau_i)} \right\rceil c(\tau_i) \leq Q$$

By plugging in the definition of the tasks, we can read this inequality in a different way, namely as

$$\frac{\varepsilon}{\delta} + \sum_{i=1}^{n} \lceil Q\alpha_i \rceil \frac{w_i}{\delta} \leq Q$$

and consequently

$$\sum_{i=1}^{n} \lceil Q\alpha_i \rceil \, w_i \leq Q\delta.$$

Now the approximation error can be bounded by

$$\sum_{i=1}^{n} w_i(\lceil Q\alpha_i \rceil - Q\alpha_i) = \underbrace{\sum_{i=1}^{n} w_i \lceil Q\alpha_i \rceil - Q \overbrace{\left[\sum_{i=1}^{n} w_i\alpha_i + \frac{4\varepsilon}{N}\right]}^{=\delta}}_{\leq 0} + 4\varepsilon \underbrace{\frac{Q}{N}}_{\leq \rho} \leq 4\rho \cdot \varepsilon$$

and the theorem follows. □

By combining Corollary 8.2 and Theorem 8.7, we conclude

**Corollary 8.8.** *There exists a universal constant $c > 0$ such that: Given a system $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$ consisting of implicit-deadline tasks, i.e. $\tau_i = (c(\tau_i), p(\tau_i))$, $0 \leq p(\tau_1) \leq \ldots < p(\tau_n)$, $0 \leq c(\tau_i) \leq p(\tau_i)$ $\forall i = 1, \ldots, n$, it is* **NP***-hard to distinguish between $r(\tau_n) \leq p(\tau_n)$ and $r(\tau_n) \geq n^{c/\log\log n} \cdot p(\tau_n)$.*

*In other words, solving*

$$\min\left\{r \geq 0 \mid c(\tau_n) + \sum_{i=1}^{n-1} \left\lceil \frac{r}{p(\tau_i)} \right\rceil c(\tau_i) \leq r\right\}$$

*is* **NP**-*hard.*

*Remark* 8.9. In this section, we showed that computing the response time of a single task in a task system is **NP**-hard, i.e. deciding whether all jobs of this particular task meet their deadlines is difficult. Formally this does not imply **NP**-hardness as well for the feasibility test, since at this point we cannot exclude that one of the prior tasks is already infeasible, thus there might be an algorithm that does not need to compute the response times for all tasks. Nevertheless, we conjecture the opposite.

**Conjecture 8.10.** *Given a system $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$ of implicit-deadline tasks it is* **NP**-*hard to decide, whether $\mathcal{S}$ is RM-schedulable.*

The instance, constructed in the reduction in Section 8.1 contains a number $\alpha_0$ with an astronomically large weight $w_0$. As a consequence, the utilization of the task system, which is build in the hardness proof for response time computation is extremely close to 1. This raises the following suspicion:

**Conjecture 8.11.** *Let $\varepsilon > 0$ be an arbitrary but fixed constant. Then the response times of a task $\tau \in \mathcal{S}$ w.r.t. an RM-schedule of an implicit-deadline task system $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$ can be computed in polynomial time if $u(\mathcal{S}) \leq 1 - \varepsilon$.*

# Chapter 9

# Intractability of Multiprocessor Scheduling

While we already obtained hardness results for single-processor scheduling, we now shed light on the complexity status of MULSCHED. Thus we are given a set $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$ of implicit-deadline tasks (i.e., $\tau_i = (c(\tau_i), p(\tau_i))$) and aim at a partition of $\mathcal{S}$ into $P_1, \ldots, P_m$ where each set $P_i$ is RM-schedulable and $m$ has to be minimized. We denote the optimum reachable $m$ as $OPT_{\text{MulSched}}$. Since MULSCHED is a generalization of BINPACKING, a reduction from PARTITION directly yields that it is **NP**-hard to distinguish between the cases, whether 2 or 3 processors are needed, hence there cannot be a polynomial time $(\frac{3}{2} - \varepsilon)$-approximation algorithm for any $\varepsilon > 0$, unless $\mathbf{NP} = \mathbf{P}$. But for BINPACKING one can obtain an *asymptotic FPTAS (AFPTAS)*, i.e. an algorithm $A$ with

$$A(\mathcal{I}) \leq (1 + \varepsilon) \cdot OPT_{\text{BinPacking}}(\mathcal{I}) + p(1/\varepsilon)$$

for a polynomial $p$, which runs in time $|\mathcal{I}|^{\mathcal{O}(1)} \cdot (1/\varepsilon)^{\mathcal{O}(1)}$. The algorithm of Karmarkar & Karp [KK82] computes even solutions with a poly-logarithmic additive gap, i.e. it respects

$$A(\mathcal{I}) - OPT_{\text{BinPacking}}(\mathcal{I}) \leq \mathcal{O}(\log^2(OPT_{\text{BinPacking}}(\mathcal{I}))).$$

Such a gap indeed respects the conditions of an AFPTAS as we may easily obtain, see e.g. [Joh92] for a more detailed account. For the sake of completeness:

**Lemma 9.1.** *An algorithm $A$ is an AFPTAS for a minimization problem if and only if there is a constant $0 < \delta < 1$ s.t.*

$$A(\mathcal{I}) - OPT(\mathcal{I}) \leq OPT(\mathcal{I})^{\delta}$$

*for all instances $\mathcal{I}$ with $OPT(\mathcal{I})$ large enough.*

*Proof.* We abbreviate $OPT := OPT(\mathcal{I})$ and suppose that $A(\mathcal{I}) - OPT \leq OPT^{\delta}$. Either one has $OPT^{\delta-1} \leq \varepsilon$, then $OPT^{\delta} = OPT^{\delta-1} \cdot OPT \leq \varepsilon \cdot OPT$, or $OPT^{\delta-1} \geq \varepsilon$, then $OPT^{\delta} = (OPT^{1-\delta})^{\delta/(1-\delta)} \leq (1/\varepsilon)^{\delta/(1-\delta)}$. In any case

$$A(\mathcal{I}) \leq (1 + \varepsilon) OPT + (1/\varepsilon)^{\delta/(1-\delta)}.$$

For the reverse direction, suppose that $A(\mathcal{I}) \leq (1+\varepsilon) \cdot OPT + p(1/\varepsilon)$ for some polynomial $p$. We assume w.l.o.g. that $p(1/\varepsilon) = \varepsilon^{-\alpha}$ for a fixed exponent $\alpha > 0$. Then for a choice of $\varepsilon := (1/OPT)^{1/(2\alpha)}$ the algorithm computes a solution with

$$A(\mathcal{I}) - OPT \leq \varepsilon \cdot OPT + (1/\varepsilon)^{\alpha} \leq OPT^{1-1/(2\alpha)} + OPT^{1/2}.$$

$\square$

In this chapter, we exclude the existence of an AFPTAS for MulSched, unless **P** = **NP**, reproducing a result from Eisenbrand & Rothvoß [ER08a]. This shows that MulSched is strictly harder than its special case BinPacking.

The hardness reduction from Partition to BinPacking yields only weak **NP**-hardness, i.e. the intractability is due to huge numbers in the instance and the optimum solution assigns many items to each bin. But it was already shown by Garey & Johnson [GJ79] that even a very restricted variant of BinPacking, called 3-Partition where exactly 3 items may be assigned per bin, is strongly **NP**-hard.

3-Partition
Given: A multi-set of $3n$ numbers $\mathcal{I} = \{a_1, \dots, a_{3n}\}$ with $\frac{1}{2} < a_i < \frac{1}{4}$ for all $i = 1, \dots, 3n$.
Decide: Is there a partition of the numbers into triples, such that the sum of the numbers of each triple is exactly one.
In other words: Is $OPT_{\text{BinPacking}}(\mathcal{I}) \leq n$.

We now derive that the **NP**-hard additive gap of 1 for 3-Partition can be transformed into an arbitrarily large additive gap for MulSched.

**Theorem 9.2.** *If* **P** $\neq$ **NP**, *there is no $\varepsilon > 0$ such that there exists a polynomial time* MulSched *algorithm, which computes an approximate solution $A(\mathcal{S})$ for each instance $\mathcal{S}$ with*

$$A(\mathcal{S}) - OPT_{\text{MulSched}}(\mathcal{S}) \leq |\mathcal{S}|^{1-\varepsilon}.$$

*This holds even if the numbers $c(\tau), p(\tau)$ in the instance are unary encoded.*

*Proof.* Suppose we are given a 3-Partition instance $\mathcal{I} = \{a_1, \dots, a_{3n}\}$ with $1/2 < a_i < 1/4$ for all $i$, where each $a_i$ is bounded by a polynomial in $n$. Let $M$ be a sufficiently large number, which we will determine later. We define a new instance $\mathcal{I}' = (a'_1, \dots, a'_{3n})$ with weights $a'_i := \frac{M/3 + a_i}{M+1}$. Then a triple in $\mathcal{I}$ sums up to at most 1 if and only if the same holds for the corresponding triple in $\mathcal{I}'$, as we can read from

$$\sum_{i \in J} a'_i = \frac{M + \sum_{i \in J} a_i}{M+1}$$

for any $J \subseteq \{1, \dots, 3n\}$ with $|J| = 3$. Hence $\mathcal{I} \in$ 3-Partition if and only if $\mathcal{I}' \in$ 3-Partition. The new instance $\mathcal{I}'$ has the advantage that $|1/3 - a'_i| < 1/M$, thus we may assume $a'_i$ to be arbitrarily close to $1/3$.

Let $k$ be an integer value that is polynomially bounded in $n$. We will show, how to blow up the **NP**-hard (additive) gap of 1 for 3-Partition to a gap of $k/2$. Define periods

$p_j = 1 + j/(4 \cdot k)$ for $j = 1, \ldots, k$. Those periods are between $1 + 1/(4 \cdot k)$ and $5/4$. The MulSched instance consists of $3kn$ many implicit-deadline tasks, which are defined as $\mathcal{S} = \{(a'_i \cdot p_j, p_j) \mid i = 1, \ldots, 3n;\ j = 1, \ldots, k\}$, using again notation $\tau = (c(\tau), p(\tau))$. Observe that the utilization of task $(a'_i \cdot p_j, p_j)$ is precisely the number $a'_i \in [0, 1]$. All tasks with period $p_j$ are called *group* $\mathcal{S}_j$. We claim that

- (1) $\mathcal{I}' \in$ 3-Partition $\Rightarrow OPT_{\text{MulSched}}(\mathcal{S}) \leq nk$
- (2) $\mathcal{I}' \notin$ 3-Partition $\Rightarrow OPT_{\text{MulSched}}(\mathcal{S}) \geq nk + k/2$

Once we have obtained this, we are done, since for any $0 < \varepsilon < 1$, we may then choose $k := n^{1/(1-\varepsilon)}$ and the gap is

$$k/2 = \frac{1}{2} n^{1/(1-\varepsilon)} > 3n^{(1+\frac{1}{1-\varepsilon})\cdot\varepsilon} \geq (3kn)^{\varepsilon} = |\mathcal{S}|^{\varepsilon}$$

for $n$ large enough.

We start by showing claim (1). Given a partition of the numbers in $\mathcal{I}'$, such that the sum of each triple is at most 1. Then by applying the same partition to each group we obtain triples of tasks, whose utilization is at most 1. Since the periods in each group are the same, all triples of tasks are then RM-schedulable (cf. Chapter 2).

Case (2) is the more challenging one. First we show that 3 tasks are not RM-schedulable on a single processor, if they are not all stemming from the same group. For this aim choose tasks $\mathcal{S}' = \{\tau_1, \tau_2, \tau_3\} \subseteq \mathcal{S}$ with $\tau_1 \prec \tau_2 \prec \tau_3$, thus $p(\tau_1) \leq p(\tau_2) \leq p(\tau_3)$. Since not all are from the same group, we have $p(\tau_1) < p(\tau_3)$. We argue that $\tau_3$ is infeasible, if $\mathcal{S}'$ is scheduled on one processor using Rate-monotonic scheduling. Let $r$ be the response time of $\tau_3$, then

$$r = c(\tau_3) + \left\lceil \frac{r}{p(\tau_1)} \right\rceil c(\tau_1) + \left\lceil \frac{r}{p(\tau_2)} \right\rceil c(\tau_2). \tag{9.1}$$

Ignoring the rounding operations yields

$$r \geq c(\tau_3) + r \cdot u(\tau_1) + r \cdot u(\tau_2)$$

and consequently

$$r \geq \frac{c(\tau_3)}{1 - u(\{\tau_1, \tau_2\})} \geq p(\tau_3) \cdot \frac{1/3 - 1/M}{1 - 2 \cdot (1/3 - 1/M)} = p(\tau_3) \cdot \frac{M-3}{M+6}$$

Notice that $p(\tau_3) \geq p(\tau_1) + \frac{1}{4k}$, thus

$$\frac{r}{p(\tau_1)} \geq \frac{M-3}{M+6} \cdot \frac{p(\tau_3)}{p(\tau_1)} \geq \frac{M-3}{M+6} \cdot \frac{p(\tau_1) + \frac{1}{4k}}{\underbrace{p(\tau_1)}_{\leq 5/4}} \overset{M:=90k}{\geq} \frac{90k-3}{90k+6} \cdot \left(1 + \frac{1}{5k}\right) > 1$$

Hence

$$r = c(\tau_3) + \underbrace{\left\lceil \frac{r}{p(\tau_1)} \right\rceil}_{\geq 2} c(\tau_1) + \left\lceil \frac{r}{p(\tau_2)} \right\rceil c(\tau_2) \geq 4 \cdot \left(\frac{1}{3} - \frac{1}{M}\right) > \frac{5}{4} \geq p(\tau_3)$$

and task $\tau_3$ must be infeasible.

It remains to account the number of necessary processors. Let $m_i$ be the number of processors, which contain 3 tasks from group $\mathcal{S}_i$. The remaining $3nk - \sum_{i=1}^{k} 3m_i$ many tasks need at least $\frac{1}{2}(3nk - 3\sum_{i=1}^{k} m_i)$ many processors. Recalling that $m_i \leq n-1$, we conclude that one needs at least

$$\sum_{i=1}^{k} m_i + \frac{1}{2}(3nk - 3\sum_{i=1}^{k} m_i) = \frac{3}{2}nk - \frac{1}{2}\sum_{i=1}^{k} m_i \geq \frac{3}{2}nk - \frac{1}{2}k(n-1) = nk + k/2$$

many processors. Finally the choice of $M$ was modest enough, s.t. the numbers in instance $\mathcal{S}$ are still polynomially bounded in $n$. The claim then follows. □

**Corollary 9.3.** *There is no asymptotic FPTAS for* MulSched*, unless* $\mathbf{P} = \mathbf{NP}$*. This holds even if the numbers* $c(\tau_i), p(\tau_i)$ *are unary encoded.*

We obtained that MulSched is harder to approximate that BinPacking. On the other hand in Chapter 3 we derived the existence of an asymptotic PTAS under resource augmentation, i.e. a PTAS where the processors are allowed to be slightly infeasible. Thus our results leave a gap and the following question arises:

**Open Problem 9.4.** *Does there exist an asymptotic PTAS for* MulSched*?*

To disprove the existence of such an approximation scheme, it might be needed to incorporate problems for which much stronger inapproximability results are possible that are based on the *PCP theorem* [ALM$^+$92, ALM$^+$98]. Note that the hardness proof of 3-Partition does not make use of this powerful tool.

# Chapter 10

# Fixed Priorities vs. Dynamic Priorities

A set of implicit-deadlines tasks is RM-schedulable if their utilization does not exceed $\ln(2) \approx 0.69$, while for any $\varepsilon > 0$ one can find a task system with utilization $\ln(2) + \varepsilon$, which is not RM-schedulable [LL73]. On the other hand, if the scheduling algorithm may assign priorities that depend on time, then the optimal policy is *Earliest Deadline First* and all tasks meet their deadlines if and only if the utilization is bounded by 1.

In other words, if we are given an EDF-schedulable implicit-deadline task system, then in the worst case, we need a processor which is $1/\ln(2) \approx 1.44$ times faster, to achieve RM-schedulability.

This gives an exact quantification of the power of fixed priority scheduling compared to dynamic priorities — at least if we talk about tasks with implicit deadlines. But how is the situation for constrained deadlines?

Suppose we are given $n$ tasks $\tau_1, \ldots, \tau_n$, where now each task $\tau$ is described by running time $c(\tau)$, period $p(\tau)$ and additionally an explicit, but constrained deadline $d(\tau) \leq p(\tau)$. To state a feasibility condition for EDF, first recall the *demand bound function*, giving the cumulated running time of all tasks, whose release time and deadline lie in $[0, t]$.

$$\texttt{DBF}(\mathcal{S}, t) = \sum_{\tau \in \mathcal{S}} \left( \left\lfloor \frac{t - d(\tau)}{p(\tau)} \right\rfloor + 1 \right) \cdot c(\tau)$$

For feasibility it is of course necessary that $\texttt{DBF}(\mathcal{S}, t) \leq t$ for all $t \geq 0$. But this condition is also sufficient, see [BMR90, BRH90]. Furthermore recall the *load*

$$\texttt{LOAD}(\mathcal{S}) = \max_{t>0} \left\{ \frac{\texttt{DBF}(\mathcal{S}, t)}{t} \right\}$$

as the maximum utilization in any interval $]0, t]$. Note that this maximum is always attained. Then $\mathcal{S}$ is EDF-schedulable if and only if $\texttt{LOAD}(\mathcal{S}) \leq 1$ [BMR90, BRH90].

Among the static-priority policies *deadline monotonic* scheduling is optimal for such constrained-deadline systems, meaning that the priority of a task $\tau$ is higher for smaller $d(\tau)$ [LW82].

We call a task *DM-tight*, if any increase of its running time would cause a deadline miss in a DM-schedule. Similar a task system is DM-tight if it contains at least one task, which is DM-tight. It might lack intuition but we do not assume that a tight schedule is feasible. This prevents that we waste time and space in the proofs for showing a property that is not needed for the main result. Thus a task system that is not DM-schedulable is always DM-tight, but not necessarily vice versa. The precise question that we are going to answer here is

*What is the minimum value $\alpha_n$, such that for any $\varepsilon > 0$, there is a constrained-deadline system consisting of $n$ tasks, with load $\alpha_n + \varepsilon$, which is not DM-schedulable.*

We can also state it as

$$\alpha_n \; := \; \inf \{\texttt{LOAD}(\mathcal{S}) \mid \mathcal{S} \text{ not DM-schedulable}, |\mathcal{S}| = n\}$$

By either splitting a task $\tau$ into two tasks $\tau', \tau''$ with the same deadline and period as $\tau$ but $c(\tau') + c(\tau'') = c(\tau)$ or by adding tasks with zero running time, we see that $\alpha_n$ is a monotone non-increasing sequence. Thus the limit $\alpha := \lim_{n\to\infty} \alpha_n$ is well-defined. Of course $0 \leq \alpha \leq \ln(2) \approx 0.69$, as it is witnessed by the implicit-deadline task system from Liu & Layland [LL73] since in case of $d(\tau) = p(\tau)$ for all tasks, one has $\texttt{LOAD}(\mathcal{S}) = u(\mathcal{S})$ and DM-scheduling coincides with RM-scheduling. Moreover it was shown by Baruah & Burns [BB08] that $1/2 \leq \alpha \leq 2/3$. The main result from this chapter will be to pinpoint this constant to $\alpha = \Omega \approx 0.5671$, where $\Omega$ is a constant, which usually appears in complex calculus. It is uniquely defined as solution of the equation $x \cdot e^x = 1$ for $x \in \mathbb{R}_+$ or alternatively by $x = \ln(1/x)$. In other words we obtain that, given an EDF-schedulable constrained-deadline task system, the system is also DM-schedulable if the unit-speed processor is replaced by a processor of speed $1/\Omega \approx 1.7632$. We will see that this value is also tight. Note that this chapter is borrowed from Davis, Rothvoß, Baruah & Burns [DRBB09].

But under which conditions is a task system DM-schedulable? Order tasks such that $\tau_1 \prec \ldots \prec \tau_n$, i.e. $d(\tau_1) \leq \ldots \leq d(\tau_n)$, and consider the interval $[0, r]$, where $r$ gives the response time for task $\tau_i$. Of course in this interval we have to execute $\tau_i$. But any task $\tau_j$ with $j < i$ will be released and executed exactly $\lceil r/p(\tau_j) \rceil$ times, consuming time $c(\tau_j)$ for each execution. Thus it is not surprising that $\tau_i$ meets all deadlines if and only if

$$\exists 0 \leq r \leq d(\tau_i) : c(\tau_i) + \sum_{j<i} \left\lceil \frac{r}{p(\tau_j)} \right\rceil \cdot c(\tau_j) \leq r$$

see [LSD89]. This condition is similar to RM-schedulability, with the slight difference that the upper bound on the response time is now $d(\tau_i)$ instead of $p(\tau_i)$.

By basically negating the above condition, we see that $\tau_i$ is DM-tight if and only if

$$\forall 0 \leq r \leq d(\tau_i) : c(\tau_i) + \sum_{j<i} \left\lceil \frac{r}{p(\tau_j)} \right\rceil \cdot c(\tau_j) \geq r.$$

The left hand side expression is termed the *DM-demand* of $\tau_i$ w.r.t. $r$.

## 10.1 Characterization of DM-tight systems

The following theorem states that among all possible sequences of task systems that determine the infimum for $\alpha_n$, there must be at least one sequence whose task systems are of a very special form. The proof itself consist of a series of observations.

**Theorem 10.1.** *Given any DM-tight system $\mathcal{S}$ of constrained-deadline tasks, there is another DM-tight system $\mathcal{S}' = \{\tau_1, \ldots, \tau_m\}$, such that $m \leq |\mathcal{S}|$, $\texttt{LOAD}(\mathcal{S}') \leq \texttt{LOAD}(\mathcal{S})$ and the following properties hold*

- $1/2 < d(\tau_1) \leq \ldots \leq d(\tau_{m-1}) < d(\tau_m) = 1$
- $p(\tau_i) = d(\tau_i) \quad \forall i = 1, \ldots, m-1$
- $p(\tau_m) = \infty$
- *Task $p(\tau_m)$ is DM-tight*

*Proof.* Consider a task system $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$ which is DM-tight. Let $d(\tau_1) \leq \ldots \leq d(\tau_n)$. By definition there must be task, say $\tau_i$, which is DM-tight. Removing the lower priority tasks $\tau_{i+1}, \ldots, \tau_n$ may only decrease the load, while not affecting DM-tightness of $\tau_i$. Thus we may assume that it is the lowest priority task $\tau_n$ which is tight. DM-tightness and the load are invariant under scaling, thus we may further assume that $d(\tau_1) \leq \ldots \leq d(\tau_n) = 1$. After plugging in the definition of the load and DM-tightness we obtain a not very handy formula for $\alpha_n$

$$\begin{aligned}
&\inf_{|\mathcal{S}|=n} \{\texttt{LOAD}(\mathcal{S}) \mid \tau_n \text{ DM-tight}\} \\
= &\inf_{|\mathcal{S}|=n} \left\{ \max_{t>0} \left\{ \frac{\sum_{i=1}^{n} \left( \left\lfloor \frac{t-d(\tau_i)}{p(\tau_i)} \right\rfloor + 1 \right) c(\tau_i)}{t} \right\} \mid \forall 0 \leq r \leq 1 : c(\tau_n) + \sum_{i<n} \left\lceil \frac{r}{p(\tau_i)} \right\rceil c(\tau_i) \geq r \right\}
\end{aligned}$$

Observe that $p(\tau_n)$ only appears in the expression for $\texttt{LOAD}(\mathcal{S})$ on the left hand side. Furthermore increasing it can only lower the infimum. Thus we may choose $p(\tau_n) = \infty$. Suppose there is an $i < n$ with $p(\tau_i) \geq 1$. Defining $d(\tau_i) := 1$ can only lower the load. Note that for $r \in ]0, 1]$ we have $\lceil r/p(\tau_i) \rceil = 1$. This does not change, when sending $p(\tau_i)$ to $\infty$ and the load decreases. But $(t - d(\tau_i))/p(\tau_i) \to 0$, thus $\texttt{LOAD}(\mathcal{S})$ decreases. As a consequence we might obtain a subset $\mathcal{S}^*$ of tasks with $d(\tau) = 1$ and $p(\tau) = \infty$ for all $\tau \in \mathcal{S}^*$. If $\mathcal{S}^* \neq \emptyset$, then we replace the tasks in $\mathcal{S}^*$ by a super task $\tau^* = (c(\tau^*), d(\tau^*), p(\tau^*)) = (\sum_{\tau \in \mathcal{S}^*} c(\tau), 1, \infty)$ without affecting DM-tightness or load. From now on we may assume that $p(\tau_i) < 1$ for all $i < n$. Next observe that also $d(\tau_i)$ for $i \in \{1, \ldots, n-1\}$ appears only in the expression for $\texttt{LOAD}(\mathcal{S})$, not in the DM-tightness constraint. Again increasing each $d(\tau_i)$ as much as possible can only decrease $\texttt{LOAD}(\mathcal{S})$. Since we are restricted to constrained-deadline tasks, the maximum admissible value for $d(\tau_i)$ (for $i < n$) is $d(\tau_i) = p(\tau_i)$.

Next introduce an auxiliary function

$$\mathbf{1} : \mathbb{R} \to \{0, 1\}, \quad \mathbf{1}(t) := \begin{cases} 1 & \text{if } t \geq 1 \\ 0 & \text{if } t < 1 \end{cases}$$

By incorporating all simplifying assumptions that we may made so far, we see that the expression for $\alpha_n$ simplifies to

$$\inf_{|\mathcal{S}|=n} \left\{ \max_{t>0} \left\{ \frac{\mathbf{1}(t) \cdot c(\tau_n) + \sum_{i<n} \left\lfloor \frac{t}{p(\tau_i)} \right\rfloor c(\tau_i)}{t} \right\} \mid \forall 0 \leq r \leq 1 : c(\tau_n) + \sum_{i<n} \left\lceil \frac{r}{p(\tau_i)} \right\rceil c(\tau_i) \geq r \right\}$$

It remains to shift the deadlines of all tasks into the interval $]1/2,1]$. To this aim, consider a task $\tau_i$ with $p(\tau_i) \leq 1/2$. Choose an integer $k \geq 2$, such that $1/2 < k \cdot p(\tau_i) \leq 1$. Then

$$\left\lceil \frac{r}{kp(\tau_i)} \right\rceil kc(\tau_i) \geq \left\lceil \frac{r}{p(\tau_i)} \right\rceil c(\tau_i)$$

and

$$\left\lfloor \frac{t}{kp(\tau_i)} \right\rfloor kc(\tau_i) \leq \left\lfloor \frac{t}{p(\tau_i)} \right\rfloor c(\tau_i)$$

Consequently replacing $\tau_i$ by $\tau_i' = (c(\tau_i'), d(\tau_i'), p(\tau_i')) = (kc(\tau_i), kd(\tau_i), kp(\tau_i))$ can only increase the DM-demand and lower the load. Tasks with deadline 1 can again be merged with $\tau_n$. We conclude that starting from a DM-tight task system, we derived a task system with not more tasks, but fulfilling the claimed properties. □

## 10.2 Lower bounding $\alpha_n$

Before we continue with bounds on $\alpha_n$, we need an improved lower bound on the utilization of a not RM-schedulable system. Recall that in general this bound is just $n(\sqrt[n]{2}-1)$. But if we know that there is at least one task with a high utilization, then we can improve this quantity. Note that for the next lemma we are back to the setting of RM-scheduling of implicit-deadline tasks.

**Lemma 10.2.** *Let $n \geq 2$. Suppose $\tau = (\tau_1, \ldots, \tau_n)$ with $\tau_i = (c(\tau_i), p(\tau_i))$ is a system of implicit-deadline tasks, which is not RM-schedulable. Then*

$$u(\mathcal{S}\backslash\{\tau_n\}) > (n-1) \cdot \left( \sqrt[n-1]{\frac{2}{1+u(\tau_n)}} - 1 \right) \geq \ln\left(\frac{2}{1+u(\tau_n)}\right)$$

*Proof.* We begin by showing the first inequality. Since $\mathcal{S}$ is not RM-schedulable, according to the hyperbolic bound of Bini & Buttazzo [BBB01] we must have

$$\prod_{i=1}^{n}(1+u(\tau_i)) > 2$$

Then

$$\begin{aligned} 1 + \frac{\sum_{i=1}^{n-1} u(\tau_i)}{n-1} &= \sum_{i=1}^{n-1} \frac{1+u(\tau_i)}{n-1} \\ &\overset{(*)}{\geq} \sqrt[n-1]{\prod_{i=1}^{n-1}(1+u(\tau_i))} \\ &\overset{(**)}{>} \sqrt[n-1]{\frac{2}{1+u(\tau_n)}} \end{aligned}$$

Here in $(*)$ we use the inequality of arithmetic and geometric mean and in $(**)$ we plug in the hyperbolic bound (divided by $1+u(\tau_n)$). Rearranging terms then yields the first claimed inequality.

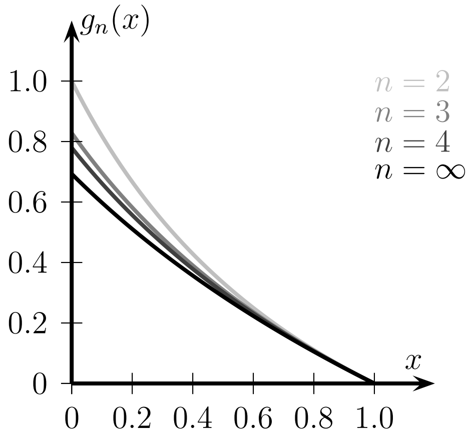


Figure 10.1: Functions $g_n(x) = (n-1) \cdot \left( \sqrt[n-1]{\frac{2}{1+x}} - 1 \right)$ with $g_\infty(x) = \ln(\frac{2}{1+x})$.

For the second part of the claim, observe that

$$(n-1) \cdot \left( \sqrt[n-1]{\frac{2}{1+u(\tau_n)}} - 1 \right) \geq (n-1) \cdot \ln \left( \sqrt[n-1]{\frac{2}{1+u(\tau_n)}} \right) = \ln \left( \frac{2}{1+u(\tau_n)} \right)$$

applying $\ln(y) \leq y - 1$ for $y > 0$. □

Note that

$$\lim_{n\to\infty} (n-1) \cdot \left( \sqrt[n-1]{\frac{2}{1+u(\tau_n)}} - 1 \right) = \ln \left( \frac{2}{1+u(\tau_n)} \right),$$

see Figure 10.1 for a visualization. If $\mathcal{S}$ is just RM-tight, then Lemma 10.2 still holds with "$\geq$" instead of strict inequality.

**Lemma 10.3.** *Given a constrained-deadline task system $\mathcal{S} = \{\tau_1, \ldots, \tau_n\}$ with*

- $1/2 < d(\tau_1) \leq \ldots \leq d(\tau_{n-1}) < d(\tau_n) = 1$
- $p(\tau_i) = d(\tau_i) \quad \forall i = 1, \ldots, n-1$
- $p(\tau_n) = \infty$
- *Task $\tau_n$ is DM-tight*

*Then*

$$\begin{aligned} \texttt{LOAD}(\mathcal{S}) &\geq \max \left\{ \frac{1+c(\tau_n)}{2}, (n-1) \cdot \left( \sqrt[n-1]{\frac{2}{1+c(\tau_n)}} - 1 \right) \right\} \\ &\geq \max \left\{ \frac{1+c(\tau_n)}{2}, \ln \left( \frac{2}{1+c(\tau_n)} \right) \right\} \end{aligned}$$

*Proof.* By evaluating $\texttt{DBF}(\mathcal{S},t)$ just at $t=1$ and $t=\infty$ we obtain

$$\texttt{LOAD}(\mathcal{S}) = \max_{t>0}\left\{\frac{\texttt{DBF}(\mathcal{S},t)}{t}\right\} \geq \max\{\texttt{DBF}(\mathcal{S},1), u(\mathcal{S})\}$$

Here we use that

$$\lim_{t\to\infty}\frac{\texttt{DBF}(\mathcal{S},t)}{t} = u(\mathcal{S}).$$

We first lower bound $\texttt{DBF}(\mathcal{S},1)$. By definition

$$\texttt{DBF}(\mathcal{S},1) = c(\tau_n) + \sum_{i=1}^{n-1}\underbrace{\left\lfloor\frac{1}{p(\tau_i)}\right\rfloor}_{=1} c(\tau_i) = \sum_{i=1}^{n} c(\tau_i)$$

since $1/2 < p(\tau_i) \leq 1$. Recall that $\mathcal{S}$ is not DM-schedulable, thus especially for $r=1$ we must have

$$c(\tau_n) + \sum_{i=1}^{n-1} 2c(\tau_i) = c(\tau_n) + \sum_{i<n}\underbrace{\left\lceil\frac{1}{p(\tau_i)}\right\rceil}_{=2} c(\tau_i) \geq 1$$

since $1/2 < p(\tau_i) < 1$ for $i < n$. Hence $\sum_{i=1}^{n-1} c(\tau_i) \geq \frac{1-c(\tau_n)}{2}$ and

$$\texttt{DBF}(\mathcal{S},1) = \sum_{i=1}^{n} c(\tau_i) \geq \frac{1-c(\tau_n)}{2} + c(\tau_n) = \frac{1+c(\tau_n)}{2}.$$

On the other hand the DM-tightness of $\tau_n$ in $\mathcal{S}$ implies that

$$(c(\tau_1), p(\tau_1)),\ \ldots,\ (c(\tau_{n-1}), p(\tau_{n-1})),\ (c(\tau_n), 1)$$

considered as an implicit-deadline system is RM-tight. Therefore

$$u(\mathcal{S}) = u(\mathcal{S}\backslash\{\tau_n\}) \geq (n-1)\cdot\left(\sqrt[n-1]{\frac{2}{1+c(\tau_n)}} - 1\right)$$

using Lemma 10.2 and the fact that $u(\tau_n) = 0$ due to $p(\tau_n) = \infty$. We combine both bounds to derive

$$\texttt{LOAD}(\mathcal{S}) \geq \max\left\{\frac{1+c(\tau_n)}{2}, (n-1)\cdot\left(\sqrt[n-1]{\frac{2}{1+c(\tau_n)}} - 1\right)\right\}$$

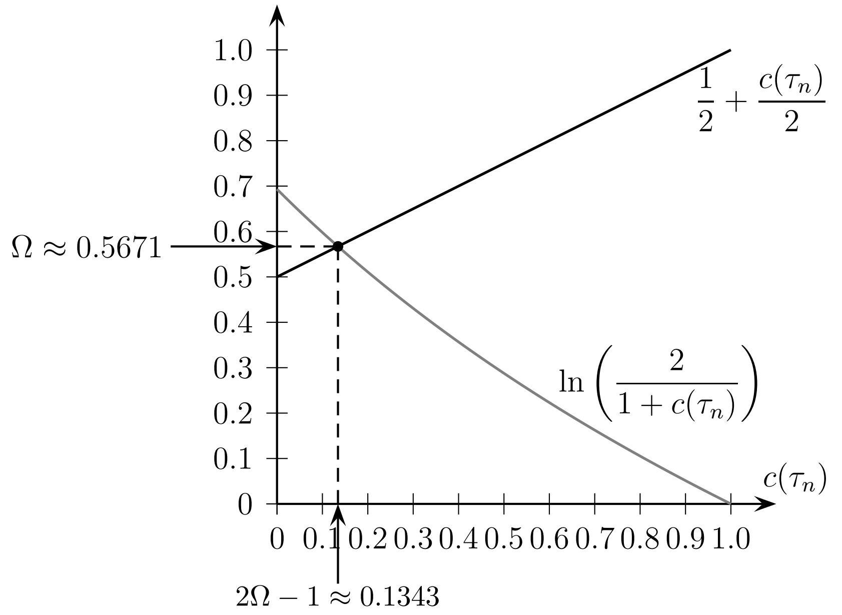
The second term in the maximum is at least $\ln(\frac{2}{1+c(\tau_n)})$ as was shown in Lemma 10.2. □

**Lemma 10.4.** *One has* $\alpha_n \geq \alpha \geq \Omega \approx 0.5671$*, where* $\Omega$ *is the unique positive real root of* $x \cdot e^x = 1$*.*

*Proof.* Let $\mathcal{S}$ a DM-tight system of constrained-deadline tasks, fulfilling the properties guaranteed by Theorem 10.1. From Lemma 10.3 we obtain

$$\texttt{LOAD}(\mathcal{S}) \geq \max\left\{\frac{1+c(\tau_n)}{2}, \ln\left(\frac{2}{1+c(\tau_n)}\right)\right\}$$

Since the left hand function is monotonic increasing with $c(\tau_n)$, the right hand function is monotonic decreasing, both functions are continuous and $\frac{1+0}{2} < \ln(2/1)$, while $\frac{1+1}{2} > \ln(2/2)$, the minimum must be attained at their unique intersection.



Recalling the definition of $\Omega$, we see that this happens for $c(\tau_n) = 2\Omega - 1 \approx 0.1343$, thus the maximum is at least $\Omega$. □

For small $n$ we can improve this lower bound. To this end define

$$\beta_n := \min_{0 \le x \le 1} \max \left\{ \frac{1+x}{2}, (n-1) \cdot \left( \sqrt[n-1]{\frac{2}{1+x}} - 1 \right) \right\}$$

Note that $\alpha_n \ge \beta_n$. Numerically one can obtain

| $n$ | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 10 | 100 | 1000 |
|---|---|---|---|---|---|---|---|---|---|---|
| $\beta_n$ | 0.6180 | 0.5943 | 0.5856 | 0.5811 | 0.5784 | 0.5766 | 0.5752 | 0.5735 | 0.5677 | 0.5672 |

where the $\beta_n$'s are rounded down to the 4th digit. See also Figure 10.2. However, while our bound is tight for $n \to \infty$, this is not the case for small $n$. In [DRBB09] it is for example shown that $\alpha_2 = \frac{1}{\sqrt{2}} \approx 0.7071$.

## 10.3 A worst-case task system

In this section, we are going to construct a family of DM-tight task systems, whose load tends to $\Omega$. This show that our above bound on $\alpha$ is tight.

How can we construct such a system? Theorem 10.1 shows already that we may choose $c(\tau_n) = 2\Omega - 1$ and $d(\tau_n) = 1$ and $p(\tau_n) = \infty$. Furthermore we must have $1/2 < d(\tau_i) = p(\tau_i) < 1$ for all other tasks $i = 1, \dots, n-1$. To complete the definition of the system we just need to come up with the idea that the running times $c(\tau_i)$ should be identical for all $i = 1, \dots, n-1$ and the deadlines $d(\tau_i)$ must just be early enough so that in the DM-schedule there is no idle time in the interval $[0, 1]$. Formally we define

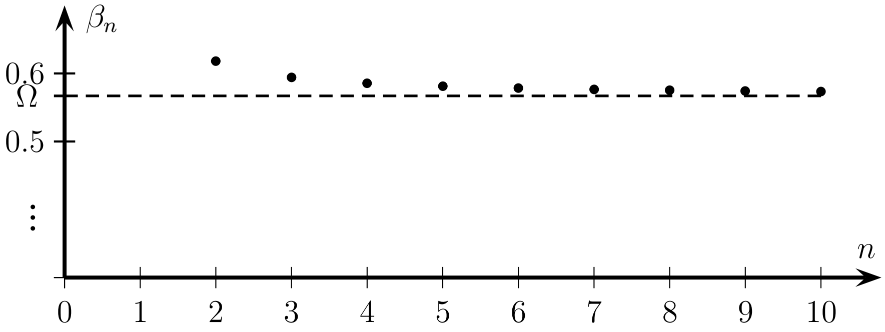
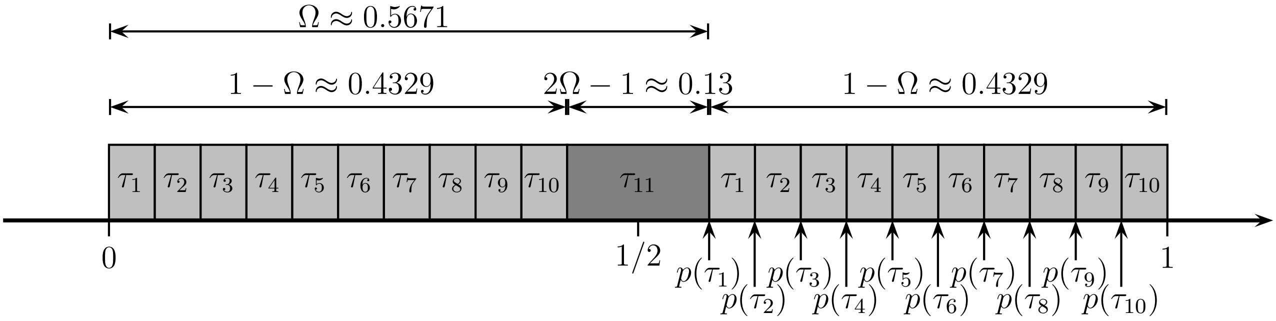


Figure 10.2: Improved lower bound $\beta_n$ on the load of DM-tight constrained-deadline task systems.



$\Omega \approx 0.5671$
$1-\Omega \approx 0.4329$
$2\Omega - 1 \approx 0.13$
$1-\Omega \approx 0.4329$
$\tau_1$ $\tau_2$ $\tau_3$ $\tau_4$ $\tau_5$ $\tau_6$ $\tau_7$ $\tau_8$ $\tau_9$ $\tau_{10}$ $\tau_{11}$ $\tau_1$ $\tau_2$ $\tau_3$ $\tau_4$ $\tau_5$ $\tau_6$ $\tau_7$ $\tau_8$ $\tau_9$ $\tau_{10}$
0
1/2
1
$p(\tau_1)$
$p(\tau_2)$
$p(\tau_3)$
$p(\tau_4)$
$p(\tau_5)$
$p(\tau_6)$
$p(\tau_7)$
$p(\tau_8)$
$p(\tau_9)$
$p(\tau_{10})$

Figure 10.3: DM-schedule for $\mathcal{S}_n$ with $n = 11$ tasks

the task system $\mathcal{S}_n = \{\tau_1, \ldots, \tau_n\}$ with

$$\begin{array}{llllllll} \tau_i: & c(\tau_i) & := & \frac{1-\Omega}{n-1} & d(\tau_i) & := & \Omega + \frac{(1-\Omega)(i-1)}{n-1} & p(\tau_i) := d(\tau_i) \quad \forall i < n \\ & & \approx & \frac{0.4329}{n-1} & & \approx & 0.5671 + 0.4329\frac{i-1}{n-1} & \\ \\ \tau_n: & c(\tau_n) & := & 2\Omega - 1 & d(\tau_n) & := & 1 & p(\tau_n) := \infty \\ & & \approx & 0.1342 & & & & \end{array}$$

**Lemma 10.5.** *$\mathcal{S}_n$ is DM-tight.*

*Proof.* Observe that indeed $d(\tau_1) < d(\tau_2) < \ldots < d(\tau_n)$. We need to show that the DM-schedule does not leave any idle time in the interval $[0, 1]$ (which is the interval of the critical instance of $\tau_n$). First of all

$$\sum_{i=1}^{n} c(\tau_i) = (2\Omega - 1) + (n-1) \cdot \frac{1-\Omega}{n-1} = \Omega$$

thus in the interval $[0, \Omega]$ the processor is fully busy with the first jobs of $\tau_1, \ldots, \tau_n$. Now consider $r \in [\Omega, 1[$. Choose $i$ such that

$$d(\tau_i) = \Omega + (1-\Omega)\frac{i-1}{n-1} \leq r < \Omega + (1-\Omega)\frac{i}{n-1} = d(\tau_{i+1})$$

If the processor is idle for a non-zero interval at time $r$, then the processor must have finished the first jobs of $\tau_1, \ldots, \tau_n$ and the second jobs of $\tau_1, \ldots, \tau_i$. But their cumulated running time is

$$\sum_{j=1}^{n} c(\tau_j) + \sum_{j=1}^{i} c(\tau_j) = \Omega + i \cdot \frac{1-\Omega}{n-1} > r$$

The contradiction yields that $\tau_n$ must be DM-tight. □

Figure 10.3 depicts the DM-schedule of $\mathcal{S}_n$. Note that in fact the DM-schedule of $\mathcal{S}_n$ is even feasible. It remains to determine the load of $\mathcal{S}_n$. However, this needs a technical analysis of several pages, which we omit here. But it can be found in [DRBB09].

**Lemma 10.6** ([DRBB09]). *One has*

$$h(t) := \lim_{n\to\infty} \mathit{DBF}(\mathcal{S}_n, t) \leq \Omega \cdot t.$$

A rough outline of this proof is as follows

1. Express $h(t)$ as a finite series.
2. Observe that $h(t)$ is piecewise linear and the slopes of consecutive pieces decrease only at $t \in \mathbb{N}$.
3. Show that $\frac{h(t+1)}{t+1} \geq \frac{h(t)}{t}$ for $t \in \{6, 7, 8, \ldots\}$.
4. Show that $\frac{h(t)}{t} \leq \Omega \; \forall t = 1, \ldots, 6$.
5. Observe that $\lim_{t\to\infty} \frac{h(t)}{t} = \lim_{n\to\infty} u(\mathcal{S}_n) = \Omega$.

From this one can then conclude that $h(t)/t \leq \Omega$. See Figure 10.4 for the graph of $h(t)/t$.

The main result of this chapter may now be summarized as

**Corollary 10.7.** *Given a constrained-deadline system $\mathcal{S}$ which is schedulable with a dynamic priority algorithm. Then it is also schedulable with a fixed-priority algorithm after increasing the speed of the processor by a factor of $1/\Omega \approx 1.7632$. This is tight in the sense that for any factor $f < 1/\Omega$ there is an EDF-schedulable constrained-deadline task system, such that a processor speed of $f$ is not sufficient for fixed-priority schedulability.*

As an open problem one might compare the power of fixed priorities and dynamic priorities if the deadlines $d(\tau)$ are arbitrary, thus if we allow $d(\tau) > p(\tau)$. There the speedup factor must lie between $1/\Omega$ and 2.

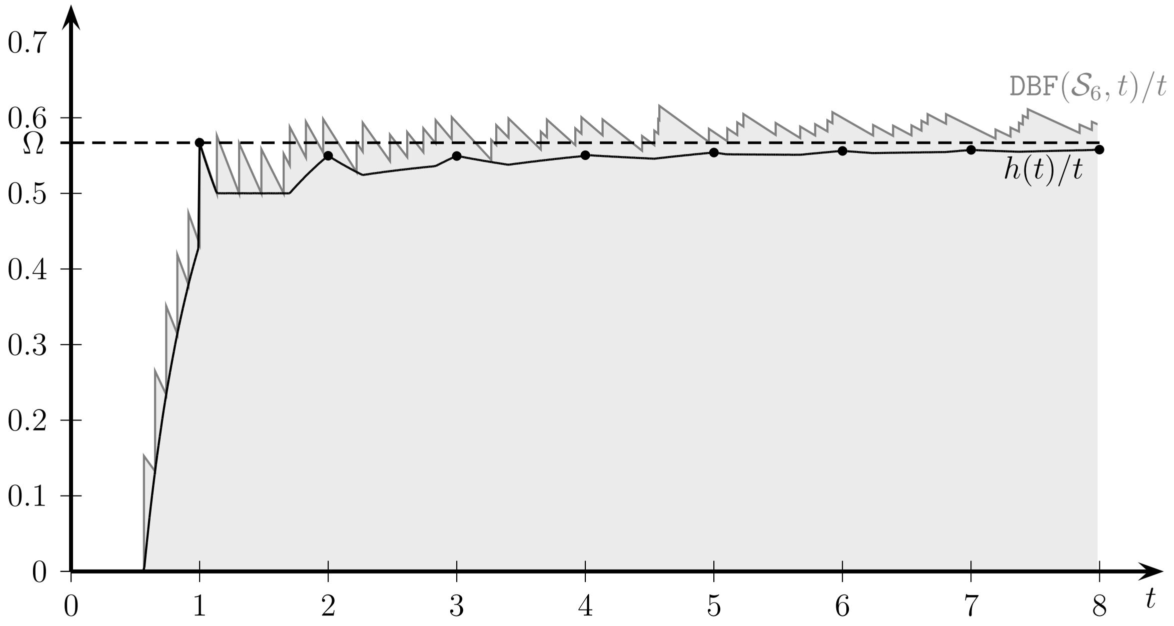


Figure 10.4: Function $\mathtt{DBF}(\mathcal{S}_6, t)/t$ in gray and $h(t)/t$ in black. Dots represent local maxima of the latter function, which appear only in case that $t$ is integer.

# Bibliography

[Ajt98] M. Ajtai. The shortest vector problem in $L_2$ is $NP$-hard for randomized reductions. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing, STOC'98 (Dallas, Texas, May 23-26, 1998)*, pages 10–19, New York, 1998. ACM Press.

[AKS01] M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*, pages 601–610 (electronic), New York, 2001. ACM.

[ALM$^+$92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *33rd Annual Symposium on Foundations of Computer Science*, pages 14–23, Pittsburgh, Pennsylvania, 24–27 October 1992. IEEE.

[ALM$^+$98] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, May 1998.

[AS04] K. Albers and F. Slomka. An event stream driven approximation for the analysis of real-time systems. In *ECRTS*, pages 187–195. IEEE Computer Society, 2004.

[AS05] K. Albers and F. Slomka. Efficient feasibility analysis for real-time systems with EDF scheduling. In *DATE*, pages 492–497. IEEE Computer Society, 2005.

[AS08] N. Alon and J. H. Spencer. *The probabilistic method*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons Inc., Hoboken, NJ, third edition, 2008. With an appendix on the life and work of Paul Erdős.

[BB08] S. K. Baruah and A. Burns. Quantifying the sub-optimality of uniprocessor fixed-priority scheduling. In *Proceedings of the 16th International Conference on Real-Time and Network Systems*. HAL - CCSD, 2008.

[BBB01] E. Bini, G. Buttazzo, and G. Buttazzo. A hyperbolic bound for the rate monotonic algorithm. In *ECRTS '01: Proceedings of the 13th Euromicro Conference on Real-Time Systems*, page 59, Washington, DC, USA, 2001. IEEE Computer Society.

[BCGM99] S. K. Baruah, D. Chen, S. Gorinsky, and A. K. Mok. Generalized multiframe tasks. *Real-Time Systems*, 17(1):5–22, 1999.

[BG04] S. K. Baruah and J. Goossens. Scheduling real-time tasks: Algorithms and complexity. In Joseph Y-T. Leung, editor, *Handbook of Scheduling — Algorithms, Models, and Performance Analysis*, chapter 28. Chapman & Hall/CRC, 2004.

[BLOS95] A. Burchard, J. Liebeherr, Y. Oh, and S. H. Son. New strategies for assigning real-time tasks to multiprocessor systems. *Institute of Electrical and Electronics Engineers. Transactions on Computers*, 44(12):1429–1442, 1995.

[BMR90] S. K. Baruah, A. K. Mok, and L. E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *IEEE Real-Time Systems Symposium*, pages 182–190, 1990.

[BP09] S. K. Baruah and K. Pruhs. Open problems in real-time scheduling. *To appear in: Journal of Scheduling*, 2009.

[BRH90] S. K. Baruah, L. E. Rosier, and R. R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, 2(4):301–324, 1990.

[CCPS97] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization.* John Wiley, New York, 1997.

[CDSW07] M. Conforti, M. Di Summa, and L. A. Wolsey. The mixing set with flows. *SIAM Journal on Discrete Mathematics*, 21(2):396–407 (electronic), 2007.

[CGJ84] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin-packing—an updated survey. In *Algorithm design for computer system design*, volume 284 of *CISM Courses and Lectures*, pages 49–106. Springer, Vienna, 1984.

[CKT02] S. Chakraborty, S. Künzli, and L. Thiele. Approximate schedulability analysis. In *IEEE Real-Time Systems Symposium*, pages 159–168, 2002.

[CM07] W. Chen and J. Meng. An improved lower bound for approximating shortest integer relation in $l_\infty$ norm (SIR$_\infty$). *Information Processing Letters*, 101(4):174–179, 2007.

[CN99] J. Cai and A. Nerurkar. Approximating the SVP to within a factor $(1 + 1/\dim^\varepsilon)$ is NP-hard under randomized reductions. *Journal of Computer and System Sciences*, 59(2):221–239, 1999. 13th Annual IEEE Conference on Computation Complexity (Buffalo, NY, 1998).

[CSHY80] Jr. Coffman, E.G., K. So, M. Hofri, and A. C. Yao. A stochastic model of bin-packing. *Information and Control*, 44(2):105–115, 1980.

[CSW08] M. Conforti, M. Di Summa, and L. A. Wolsey. The mixing set with divisible capacities. In Andrea Lodi, Alessandro Panconesi, and Giovanni Rinaldi, editors, *The 13th Conference on Integer Programming and Combinatorial Optimization, IPCO 08*, Lecture Notes in Computer Science, pages 435–449. Springer, 2008.

[CZ08] M. Conforti and G. Zambelli. The mixing set with divisible capacities: a simple approach. Manuscript, 2008.

[Der74] M. L. Dertouzos. Control robotics: The procedural control of physical processes. In *IFIP Congress*, pages 807–813, 1974.

[Dev03] U. C. Devi. An improved schedulability test for uniprocessor periodic task systems. In *ECRTS*, page 23. IEEE Computer Society, 2003.

[Dha04] S. K. Dhall. Approximation algorithms for scheduling time-critical jobs on multiprocessor systems. In Joseph Y-T. Leung, editor, *Handbook of Scheduling — Algorithms, Models, and Performance Analysis*, chapter 32. Chapman & Hall/CRC, 2004.

[Din02] I. Dinur. Approximating $\mathrm{SVP}_\infty$ to within almost-polynomial factors is NP-hard. *Theoretical Computer Science*, 285(1):55–71, 2002. Algorithms and complexity (Rome, 2000).

[Dós07] G. Dósa. The tight bound of first fit decreasing bin-packing algorithm is FFD(I) <= 11/9OPT(I) + 6/9. In Bo Chen, Mike Paterson, and Guochuan Zhang, editors, *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies, First International Symposium, ESCAPE 2007, Hangzhou, China, April 7-9, 2007, Revised Selected Papers*, volume 4614 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2007.

[DRBB09] R. Davis, T. Rothvoß, A. Burns, and S. K. Baruah. Exact quantification of the sub-optimality of uniprocessor fixed priority pre-emptive scheduling. In *To appear in Real-Time Systems*, 2009.

[Edm65a] J. Edmonds. Maximum matching and a polyhedron with $0,1$-vertices. *Journal of Research of the National Bureau of Standards*, 69B:125–130, 1965.

[Edm65b] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[ER08a] F. Eisenbrand and T. Rothvoß. A ptas for static priority real-time scheduling with resource augmentation. In *ICALP '08: Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part I*, pages 246–257, Berlin, Heidelberg, 2008. Springer-Verlag.

[ER08b] F. Eisenbrand and T. Rothvoß. Static-priority Real-time Scheduling: Response Time Computation is NP-hard. In *IEEE Real-Time Systems Symposium (RTSS)*, 2008.

[ER09] F. Eisenbrand and T. Rothvoß. New hardness results for diophantine approximation. *Accepted to APPROX'09*, 2009.

[FB05] N. Fisher and S. K. Baruah. A fully polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines. In *ECRTS '05: Proceedings of the 17th Euromicro Conference on Real-Time Systems (ECRTS'05)*, pages 117–126, Washington, DC, USA, 2005. IEEE Computer Society.

[FdlVL81] W. Fernandez de la Vega and G. S. Lueker. Bin packing can be solved within $1 + \varepsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.

[Fre80] G. N. Frederickson. Probabilistic analysis for simple one- and two-dimensional bin packing algorithms. *Information Processing Letters*, 11(4/5):156–161, 1980.

[FT87] A. Frank and É. Tardos. An application of simultaneous Diophantine approximation in combinatorial optimization. *Combinatorica*, 7:49–65, 1987.

[Gab90] H. N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'90 (San Francisco, California, January 22-24, 1990)*, pages 434–443, Philadelphia, PA, 1990. ACM SIGACT, SIAM, Society for Industrial and Applied Mathematics.

[GG61] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9:849–859, 1961.

[GGJY76] M. R. Garey, R. L. Graham, D. S. Johnson, and A. C. C. Yao. Resource constrained scheduling as generalized bin packing. *Journal of Combinatorial Theory. Series A*, 21(3):257–298, 1976.

[GGU72] M. R. Garey, R. L. Graham, and J. D. Ullman. Worst-case analysis of memory allocation algorithms. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1972.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, New York, 1979.

[GLS81] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981.

[GLS93] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2 of *Algorithms and Combinatorics.* Springer-Verlag, Berlin, second edition, 1993.

[GP01] O. Günlük and Y. Pochet. Mixing mixed-integer inequalities. *Mathematical Programming. A Publication of the Mathematical Programming Society*, 90(3, Ser. A):429–457, 2001.

[HB88] D. R. Heath-Brown. The number of primes in a short interval. *Journal für die Reine und Angewandte Mathematik*, 389:22–63, 1988.

[HBI79] D. R. Heath-Brown and H. Iwaniec. On the difference between consecutive primes. *American Mathematical Society. Bulletin. New Series*, 1(5):758–760, 1979.

[HR07] I. Haviv and O. Regev. Tensor-based hardness of the shortest vector problem to within almost polynomial factors. In *STOC'07—Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 469–477. ACM, New York, 2007.

[HW97] M. Henk and R. Weismantel. Test sets of the knapsack problem and simultaneous Diophantine approximation. In *Algorithms—ESA '97 (Graz)*, volume 1284 of *Lecture Notes in Computer Science*, pages 271–283. Springer, Berlin, 1997.

[HW02] M. Henk and R. Weismantel. Diophantine approximations and integer points of cones. *Combinatorica*, 22(3):401–407, 2002.

[JDU$^+$74] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974.

[Joh73] D. S. Johnson. *Near-optimal bin packing algorithms.* PhD thesis, MIT, Cambridge, MA, 1973.

[Joh92] D. S. Johnson. The NP-completeness column: An ongoing guide: The tale of the second prover. *Journal of Algorithms*, 13(3):502–524, September 1992.

[Kan83] R. Kannan. Polynomial-time aggregation of integer programming problems. *Journal of the Association for Computing Machinery*, 30(1):133–145, 1983.

[Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the 16th Annual ACM Symposium on the Theory of Computing*, 1984.

[Kha79] L.G. Khachiyan. A polynomial algorithm for linear programming. *Soviet Math. Doklady*, 20:191–194, 1979. (Russian original in Doklady Akademiia Nauk SSSR, 244:1093–1096).

[Kho05] S. Khot. Hardness of approximating the shortest vector problem in lattices. *Journal of the ACM*, 52(5):789–808 (electronic), 2005.

[KK82] N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *23rd annual symposium on foundations of computer science (Chicago, Ill., 1982)*, pages 312–320. IEEE, New York, 1982.

[Knö81] W. Knödel. A bin packing algorithm with complexity $O(n \log n)$ and performance 1 in the stochastic limit. In J. Gruska and M. Chytil, editors, *Mathematical Foundations of Computer Science 1981*, volume 118 of *lncs*, pages 369–378, Štrbské Pleso, Czechoslovakia, 31 August–4 September 1981. Springer-Verlag.

[KPP04] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems.* Springer, 1 edition, October 2004.

[KR09] A. Karrenbauer and T. Rothvoß. An average-case analysis for rate-monotonic multiprocessor real-time scheduling. In *Accepted for ESA'09*, 2009.

[Lag85] J. C. Lagarias. The computational complexity of simultaneous Diophantine approximation problems. *SIAM Journal on Computing*, 14(1):196–209, 1985.

[Leh90] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *IEEE Real-Time Systems Symposium*, pages 201–213, 1990.

[Len83] H. W. Lenstra, Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

[Leu04] J. Leung. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis.* CRC Press, Inc., Boca Raton, FL, USA, 2004.

[Liu00] J. Liu. *Real-Time Systems.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[LL73] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20:46–61, 1973.

[LLL82] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[LO85] J. C. Lagarias and A. M. Odlyzko. Solving low-density subset sum problems. *Journal of the Association for Computing Machinery*, 32(1):229–246, 1985.

[LS89] F. T. Leighton and P. Shor. Tight bounds for minimax grid matching with applications to the average case analysis of algorithms. *Combinatorica*, 9:161–187, 1989.

[LSD89] J. P. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *IEEE Real-Time Systems Symposium*, pages 166–171, 1989.

[Lue82] G. S. Lueker. An average-case analysis of bin packing with uniformly distributed item sizes. Technical Report 181, Dept. Information and Computer Science, University of California at Irvine, 1982.

[LW82] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250, 1982.

[MA78] K. L. Manders and L. Adleman. NP-complete decision problems for binary quadratics. *Journal of Computer and System Sciences*, 16(2):168–184, 1978.

[Mic01] D. Micciancio. The shortest vector in a lattice is hard to approximate to within some constant. *SIAM Journal on Computing*, 30(6):2008–2035 (electronic), 2001.

[MU05] M. Mitzenmacher and E. Upfal. *Probability and computing.* Cambridge University Press, Cambridge, 2005. Randomized algorithms and probabilistic analysis.

[MW03] A. J. Miller and L. A. Wolsey. Tight formulations for some simple mixed integer programs and convex objective integer programs. *Mathematical Programming. A Publication of the Mathematical Programming Society*, 98(1-3, Ser. B):73–88, 2003. Integer programming (Pittsburgh, PA, 2002).

[NZM91] I. Niven, H. S. Zuckerman, and H. L. Montgomery. *An introduction to the theory of numbers.* John Wiley & Sons Inc., New York, fifth edition, 1991.

[PW06] Y. Pochet and L. A. Wolsey. *Production planning by mixed integer programming.* Springer Series in Operations Research and Financial Engineering. Springer, New York, 2006.

[Reg04] O. Regev. New lattice-based cryptographic constructions. *Journal of the ACM*, 51(6):899–942 (electronic), 2004.

[RG96] R. Ravi and M. X. Goemans. The constrained minimum spanning tree problem (extended abstract). In *SWAT '96: Proceedings of the 5th Scandinavian Workshop on Algorithm Theory*, pages 66–75, London, UK, 1996. Springer-Verlag.

[RR06] O. Regev and R. Rosen. Lattice problems and norm embeddings. In *STOC'06: Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, pages 447–456, New York, 2006. ACM.

[RS96] C. Rössner and J. P. Seifert. Approximating good simultaneous Diophantine approximations is almost NP-hard. In *Mathematical foundations of computer science 1996 (Cracow)*, volume 1113 of *Lecture Notes in Comput. Sci.*, pages 494–505. Springer, Berlin, 1996.

[Sch87] C.-P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 53(2-3):201–224, 1987.

[Sch03] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics.* Springer, 2003.

[Sho84] P. W. Shor. The average-case analysis of some on-line algorithms for bin packing. In *Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science, FOCS'84 (Singer Island, FL, October 24-26, 1984)*, pages 193–200. IEEE, IEEE, 1984.

[Vaz01] V. V. Vazirani. *Approximation algorithms.* Springer-Verlag, Berlin, 2001.

[vEBil] P. van Emde Boas. Another $NP$-complete partition problem and the complexity of computing short vectors in a lattice. Technical Report 81-04, Mathematisch Instituut, Amsterdam, 1981, April.

[Weg05] I. Wegener. *Complexity theory.* Springer-Verlag, Berlin, 2005. Exploring the limits of efficient algorithms, Translated from the German by Randall Pruim.

[ZdF08] M. Zhao and I. de Farias. The mixing-MIR set with divisible capacities. *Mathematical Programming. A Publication of the Mathematical Programming Society*, 115(1, Ser. A):73–103, 2008.

# Appendix

## Finding dense primes

We still have to show that suitable prime numbers for the reduction from $\textsc{Sir}_\infty$ to $\mathrm{SDA}_\infty$ exist and that they can be found in polynomial time. The proof follows closely [Lag85, RS96].

**Lemma 10.8.** *Let $\rho_N = 2^{n^{\mathcal{O}(1)}}$. For a vector $a \in \mathbb{Z}^n$ of encoding size $m$ one can find distinct prime numbers $p, q_1, \ldots, q_n$ as well as natural numbers $R$ and $T$ in polynomial time, such that*

1. $n \cdot \sum_{j=1}^{n} |a_j| < p^R < q_1^T < q_2^T < \ldots < q_n^T < (1 + \frac{1}{n}) \cdot q_1^T$
2. *$p$ and all $q_i$ are co-prime to all $a_j$*
3. $q_1^T > \rho_N^2 \cdot p^R$
4. *$T, R, p, q_1, \ldots, q_n$ are bounded by a polynomial in $m$.*

*Proof.* The number of different prime factors appearing in some $a_j$ is clearly bounded by $m$. Note that $m \geq n$. Due to the prime number theorem, see, e.g. [NZM91], the first $m+1$ prime numbers can be computed in polynomial time by testing the first $\mathcal{O}(m \log(m))$ natural numbers. Choose $p$ among these primes, s.t. $\gcd(p, a_j) = 1$ for $i = 1, \ldots, n$. Compute the smallest $R$ and $T$ (for example using binary search) such that $p^R > n \cdot \sum_{j=1}^{n} |a_j|$ as well as $2^T > \rho_N^2 \cdot p^R$ and $T \geq m$. Clearly both, $R$ and $T$ are polynomially bounded in $m$. It remains to find prime numbers $q_1, \ldots, q_n$, which are sufficiently close to each other. Fortunately, we may use a very deep result in number theory at this point.

**Theorem 10.9.** *[HBI79, HB88] For each $\delta > \frac{11}{20}$ there exists a constant $c_\delta$ such that for all $z > c_\delta$ the interval $[z, z + z^\delta]$ contains a prime.*

Consider an arbitrary $i \in \{0, \ldots, T^2 - 1\}$. Choose $\delta = \frac{3}{5}$, then for sufficiently large $m$, there is a prime between each $z := T^{20} + i(2T)^{12}$ and $z + z^\delta$ with

$$z^\delta \leq (T^{20} + i(2T)^{12})^{3/5} < ((2T)^{20})^{3/5} = (2T)^{12}.$$

Thus there must be a prime in each interval $[T^{20} + i(2T)^{12}, T^{20} + (i+1)(2T)^{12}[$. Since $T$ is polynomially bounded, we can compute $m + n + 1 \leq T^2$ distinct primes from $[T^{20}, T^{20} + T^2(2T)^{12}] \subseteq [T^{20}, T^{20} + T^{15}]$. Select $n$ of these primes, which are co-prime to

$p$ and all $a_j$ and denote them by $q_1, \ldots, q_n$. Using $1 + x \le e^x \le 1 + 2x$ for $x \in [0,1]$ we obtain

$$\frac{q_n^T}{q_1^T} \le \left(\frac{T^{20} + T^{15}}{T^{20}}\right)^T = \left(1 + \frac{1}{T^5}\right)^T \le e^{T/T^5} = e^{1/T^4} \le 1 + 2\frac{1}{T^4} < 1 + \frac{1}{n}$$

Recall that $T \ge m \ge n$. The claim then follows. □

# A generalized Chernoff bound

Here we state the postponed proof of the generalized Chernoff bound, which we used in the proof of Lemma 3.9 in Chapter 3.

**Theorem.** *Let $X_1, \ldots, X_n$ be independent random variables with $X_i \in \{0, b_i\}$ for $b_i > 0$. Consider the sum $X := a_1 X_1 + \ldots + a_n X_n$ with $a_i > 0$. Then for $\alpha := \max_i\{a_i \cdot b_i\}$ and $0 < \delta < 1$ one has*

$$\Pr[X \ge (1+\delta)E[X]] \le e^{-\frac{\delta^2}{3\alpha}E[X]}$$

*Proof.* Initially we have

$$\Pr[X \ge (1+\delta)E[X]] \;=\; \Pr[e^{tX} \ge e^{t(1+\delta)E[X]}] \le \frac{E[e^{tX}]}{e^{t(1+\delta)E[X]}}$$

whereby we first used monotonicity of $f(x) = e^{tx}$ for all $t > 0$ and Markov inequality as second. Denote $\Pr[X_i = b_i] = p_i$. Multiplicativeness of the expectation of independent random variables then gives

$$\begin{aligned} E[e^{tX}] &= E[e^{\sum_{i=1}^n t a_i X_i}] \qquad (10.1)\\ &= \prod_{i=1}^n E[e^{t a_i X_i}] \\ &= \prod_{i=1}^n (p_i e^{t a_i b_i} + (1-p_i)e^{t a_i 0}) \\ &= \prod_{i=1}^n (1 + p_i(e^{t a_i b_i} - 1)) \\ &\overset{1+x\le e^x}{\le} \prod_{i=1}^n e^{p_i(e^{t a_i b_i} - 1)} \end{aligned}$$

Now choose $t = \ln((1+\delta)^{1/\alpha})$. Then (10.1) equals

$$\prod_{i=1}^n \exp\left(p_i((1+\delta)^{\frac{a_i b_i}{\alpha}} - 1)\right) \qquad (10.2)$$

Now Bernoulli inequality $\forall \delta > -1 : \forall x \in [0,1] : (1+\delta)^x \le 1 + \delta x$ can be applied since

$\frac{a_i b_i}{\alpha} \leq 1$. Thus (10.2) can be bounded by

$$\begin{aligned} \prod_{i=1}^{n} \exp\left(p_i \left(1 + \delta \frac{a_i b_i}{\alpha} - 1\right)\right) &= \prod_{i=1}^{n} \exp\left(p_i \delta \frac{a_i b_i}{\alpha}\right) \\ &= e^{\frac{\delta}{\alpha} \sum_{i=1}^{n} p_i a_i b_i} \\ &= e^{\frac{\delta}{\alpha} E[X]} \end{aligned}$$

Thereby recall that $E[X] = \sum_{i=1}^{n} a_i \cdot E[X_i] = \sum_{i=1}^{n} a_i \cdot p_i b_i$. We continue the bound on $\Pr[X \geq (1+\delta)E[X]]$ (equation (10.1))

$$\begin{aligned} \frac{E[e^{tX}]}{e^{t(1+\delta)E[X]}} &\leq \frac{e^{\frac{\delta}{\alpha} E[X]}}{(1+\delta)^{\frac{1+\delta}{\alpha} E[X]}} \\ &= \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{E[X]/\alpha} \\ &\leq e^{-\frac{\delta^2}{3\alpha} E[X]} \end{aligned}$$

whereby inequality

$$\frac{e^{\delta}}{(1+\delta)^{1+\delta}} \leq e^{-\delta^2/3}$$

comes from an elementary calculus, see, e.g. [MU05] and holds for all $0 \leq \delta \leq 1$. □

The ordinary Chernoff bound is contained as special case for $a_i = b_i = \alpha = 1$. It is possible to define $a_i' := a_i b_i$ and $b_i' := 1$. Then $X_i'$ could be assumed to be 0/1-distributed. It is not difficult to see that this is an equivalent view, since $a_i X_i$ is either 0 or $a_i b_i$, but the same holds for $a_i' X_i'$.

Other variants and many applications of the Chernoff bound can be found in the excellent book of Mitzenmacher & Upfal [MU05].

# Curriculum Vitae

| | |
|---|---|
| **Name:** | Thomas Rothvoß |
| **Nationality:** | German |
| **Date of birth:** | July 7, 1981 |
| **Place of birth:** | Essen, Germany |
| **Marital status:** | Single |
| **Address:** | Chemin du Pontet 1,<br>1037 Etagnières,<br>Switzerland |

## Biography

| | | | |
|---|---|---|---|
| 3/2008 | - | now | Research assistant at EPFL, Lausanne (Switzerland) |
| 10/2006 | - | 3/2008 | PhD student at University of Paderborn (Germany)<br>(Wissenschaftlicher Mitarbeiter) |
| 10/2002 | - | 10/2006 | Studies of Computer Science at TU Dortmund (Germany)<br>Master's thesis: "Algorithms for Virtual Private Networks"<br>Supervisors: Prof. Dr. Friedrich Eisenbrand & Dr. Piotr Krysta<br>(Final grade 1.0, summa cum laude) |
| 10/2005 | - | 10/2006 | Studies of Mathematics at TU Dortmund (Germany) |
| 1/2006 | - | 4/2006 | Student assistant (software development) at TU Dortmund |
| 4/2005 | - | 7/2005 | Student assistant for "Grundlagen der theoretischen Informatik"<br>(Foundations of theoretical computer science) at TU Dortmund |
| 7/2001 | - | 3/2002 | Mandatory military service |
| 8/1992 | - | 6/2001 | High school studies at Gymnasium Essen-Werden, (Germany) |

## Awards

- 2007: Award as *Jahrgangsbester* (valedictorian) among graduates in Computer Science 2006 at TU Dortmund

# Publications

- F. Eisenbrand, F. Grandoni, T. Rothvoß & G. Schäfer (2008): *Approximating connected facility location problems via Random facility sampling and core detouring.* ACM-SIAM Symposium on Discrete Algorithms (SODA'08), San Francisco, USA.

- F. Eisenbrand, J. Pach, T. Rothvoß & N. Sopher (2008): *Convexly independent subsets of the Minkowski sum of planar point sets.* Electronic Journal of Combinatorics, volume 15.

- F. Eisenbrand & T. Rothvoß (2008): *A PTAS for Static Priority Real-Time Scheduling with Resource Augmentation.* 35th International Colloquium on Automata, Languages and Programming (ICALP'08), Reykjavík, Iceland.

- F. Eisenbrand & T. Rothvoß (2008): *Static-priority Real-time Scheduling: Response Time Computation is NP-hard.* The Real-Time Systems Symposium (RTSS'08), Barcelona, Spain.

- F. Eisenbrand, N. Hähnle & T. Rothvoß (2009): *Diameter of Polyhedra: Limits of Abstraction.* 25th Annual Symposium on Computational Geometry (SOCG'09), Aarhus, Denmark.

- R. Davis, T. Rothvoß, S. Baruah & A. Burns (2009): *Exact quantification of the sub-optimality of uniprocessor fixed-priority pre-emptive scheduling.* Accepted for: Real-Time Systems.

- R. Cominetti, J. R. Correa, T. Rothvoß & J. San Martín (2009): *Optimal selection of customers for a last-minute offer.* Operations Research. Pending revision.

- A. Karrenbauer & T. Rothvoß (2009): *An Average-Case Analysis for Rate-Monotonic Multiprocessor Real-time Scheduling.* Accepted for: ESA'09, Copenhagen, Denmark.

- L. Sanità & T. Rothvoß (2009): *On the complexity of the asymmetric VPN problem.* Accepted for: APPROX'09, Berkeley, USA.

- F. Eisenbrand & T. Rothvoß (2009): *New Hardness Results for Diophantine Approximation.* Accepted for: APPROX'09, Berkeley, USA.

# Research Talks

- 01/2007: *An improved analysis for the random sampling algorithm for Connected Facility Location*, 11th Combinatorial Optimization Workshop, Aussois, France.

- 03/2007: *How far is a facility?*, Research Seminar, Berlin, Germany.

- 09/2007: *Minimizing congestion in the VPN setting*, Workshop on Network Design, Dortmund, Germany.

- 02/2008: *A Bi-criteria PTAS for Real-time Scheduling with fixed priorities*, Workshop on Scheduling, Schloss Dagstuhl, Germany.

- 07/2008: *A Bi-Criteria PTAS for Real-time Scheduling with fixed priorities*, 35th International Colloquium on Automata, Languages and Programming (ICALP'08), Reykjavík, Iceland.

- 07/2008: *Approximating Connected Facility Location Problems via Random Facility Sampling and Core Detouring*, 3rd Workshop on Flexible Network Design, Warwick, UK.

- 09/2008: *A PTAS for Real-time Scheduling with Fixed Priorities under Resource Augmentation*, 6. Joint Operations Research Days, Lausanne, Switzerland.

- 10/2008: *Tractability of Real-time Scheduling with fixed priorities*, DFG meeting, MPI, Saarbrücken, Germany.

- 12/2008: *Static Priority Real-time Scheduling: Response Time Computation is NP-hard*, RTSS'08, Barcelona, Spain.

- 02/2009: *An asymptotic 3/2-approximation algorithm for static-priority multiprocessor scheduling of implicit deadline tasks*, Diplomanden- und Doktorandenseminar, Berlin, Germany.