The power of simple scheduling policies

The power of simple scheduling policies

Cyriel Rutten

The power of simple scheduling policies
Cyrillus Jozef Franciscus Rutten
cyrielrutten@gmail.com
Maastricht, The Netherlands, 2013

Cover design by Nathalie Gerrekens

This book was typeset by the author using LaTeX.

Published by Universitaire Pers Maastricht
ISBN 978 94 6159 205 7
Printed in the Netherlands by Datawyse

The power of simple scheduling policies

Proefschrift

ter verkrijging van de graad van doctor
aan de Universiteit Maastricht,
op gezag van Rector Magnificus,
Prof. dr. L.L.G. Soete,
volgens het besluit van het College van Decanen,
in het openbaar te verdedigen
op vrijdag 1 februari 2013 om 12:00 uur

door

Cyrillus Jozef Franciscus Rutten

UNIVERSITAIRE PERS MAASTRICHT

Promotor:
Prof. dr. ir. C.P.M. van Hoesel

Copromotor:
Dr. T. Vredeveld

Beoordelingscommissie:
Prof. dr. ir. A.M.H. Gerards (voorzitter)
Dr. T. Harks
Prof. A. Marchetti Spaccamela (La Sapienza - Università di Roma)

Dit onderzoek werd financieel mogelijk gemaakt door de Graduate School of Business and Economics (GSBE), onderdeel van Maastricht University.

In my reflection, he lives in me.

I dedicate my dissertation to my father,
Wim
whom I wished would have been here with me,
and could have seen how my sister and me
have grown up and embraced life,

I am only me since you are you.
Thank you for being who you are.

and to,
Toos, Désirée, and
my love Josien.

Acknowledgements

These last four years have been an exciting journey taking me to different places and cities, in front of classes and through many intellectual challenges and revelations. I will memorize all those years however dearest by the people whom I met and who were at my side. I am grateful to you all for your encouragements, love and good times.

The first steps leading up to my travel led from my hometown Gulpen to Maastricht and were taken some eight years ago. I went to study the revealing science of econometrics. I was guided how to find my way through huge data sets and how to make decisions when the number of actions you can possibly take is just too overwhelming. Luckily, I had some good teachers to guide me the way. Especially Tjark, Stan, Marc, Rudolf and Alex got me excited for travelling towards the land of Operations Research.

Once there, I climbed up the highest stairs and found my own new room at the fourth floor. Stan granted me 4 years of phd-ship and a nice desk right between my fellow goal seekers. Over there we devoted our lives to science provided that it was neither too early, nor coffee-break-time. Such a life can be hard when the Utrecht train leaves way before sunrise, when hordes of students excitingly await your tale how the world can be modeled as a linear program or when an anonymous email misunderstood the significance of your contribution to that beachside conference.

I am lucky to have a supervisor who backed me up in such times and who occasionally absorbed part of my frustration as well. More often however we shared our enthusiasm and joy. Tjark, I am grateful for you being my supervisor and supporting me even though our (or mine?) headstrongness did not always align. Moreover, I am happy you send me to work with different people and on different projects.

It were my co-authors who took my mind for a stroll, or even for an exciting flight, along different problems, ideas and literature. I enjoyed discussing and cooperating with you all to reach new insights and find new theorems. Thank you, Diego, Bas, Ruben, Tobias, Suzanne, Andreas, Heiko and Nikhil. Alberto, when I think of Roman and its cuisine, I think of the special dinner you served us. I am still trying to find those nice after-dinner sweets back home. Finally, at the end of my academic journey, is the committee who read or will read the remainder of my booklet. I appreciate the time you take for reading my time spending of the last four years and for visiting Maastricht in the midst of winter.

The most memorable parts are by the people who shared in the adventure. Bas, being my best friend in university we had some good and close times together. I never forget our soul

sharing week in that purple plastic Etap cell. Ruben, together we were able to escape the ciders and find some true English ale. Thanks to you I am spending every night in Westeros for the next few years. I am happy that you both are willing to dress up on the first of February. I also thank Nathalie for designing the arty cover of my booklet which I had hoped for.

Besides, my two paranimphs, there was Diego the traveler who saved me when I was stopped by the Mexican police. I will never forget Natalya who took our discussions and storytelling beyond the Volga where the true heart of mathematics lies. Also from a far land to my east are the small and the tall, or rather the experienced observer and the always objective commentator. Norbert and Greg, you are the best in bringing some Budapest humor to Maastricht. Martijn, if wasn't for you I would have been stuck at the coffee machine instead of getting a daily fresh tea at the faculty's meeting place. I will miss the three handsome ladies there who broke the daily routine with ever new gossips and stories. And there are many more of the nicest people whom I shared a laugh with in these last four year. I am happy I got to know you and let us keep in touch!

The most fun about travelling is the home coming, family reunion, and speaking Dutch again. Bij iedere terugkomst is mijn moeder daar om me te ontvangen. Mam, toen, nu en vast nog voor vele jaren te komen sta je altijd voor me klaar met liefde en moederlijke zorg. Dankjewel voor al je kracht. Désirée en Nick, jullie zijn de gezelligste buren van Maastricht en eigenlijk vind ik het heel bijzonder dat we altijd bij elkaar binnen kunnen lopen. André, ik vind het fijn dat je bij ons hoort en altijd geïnteresseerd in ons bent. Samen met Famke en Armand schreeuw ik mee in het geel-zwarte thuisvak. Het is altijd leuk om met jullie op stap te gaan. Mijn schoonouders Jo en Evelyn wil ik graag bedanken voor alle liefde en steun vanaf de eerste vakantie tot en met taxiën naar DSM en nog vele andere fijne herinneringen.

Mijn grootste dankjewel gaat uit naar mijn vriendin Josien. Je bent mijn beste Skype maatje als ik op conferentie ben, mijn kampeergenootje in Frankrijk en mijn backpack kameraadje in Mexico. Maar nog belangrijker dan al dat, je bent de vrolijkheid bij ons thuis en mijn steun en toeverlaat met wie ik alles kan delen. Ik kijk uit naar alle avonturen die we nog samen gaan beleven!

Cyriel

Maastricht, December 2012

Contents

Chapter 1

Introduction

As the world around us grows more dynamic and data intensive everyday, optimization problems faced all over the world grow to be bigger and more complex. To deal with such large instances, there is a need for simple and powerful procedures which return good solutions fast. In this dissertation, I discuss the power of such simple policies for so-called scheduling problems. To be more explicit, I study the quality of the solutions returned by some simple and fast scheduling policies for various complex scheduling problems.

Let us consider one such complex scheduling problem which calls for a simple and good policy. Given is a computer which needs to process a set of tasks. Nowadays, computers often contain several processors and possibly these processors are even heterogeneous, that is, some processors are tailor fit to process some particular tasks while they are very slow on performing other tasks. The question arising is how to assign the different tasks to the different processors such that all tasks will be completed in time. It is easy to imagine that the flight control system of the new Airbus A380 or Joint Strike Fighter requires every few seconds the execution of a large amount of tasks like checking for altitude, speed, temperature, etc. The processor to which such a task is scheduled needs to be able to execute the task within a few seconds or even less. Checking all possible configurations for assigning computing tasks to processors to see which assignment is a good one takes too much time. In case checking a single configuration takes a millisecond, it would take almost nine thousand years to check all possible configurations when there are only 16 tasks which need to be assigned to 8 different processors. Hence, more sophisticated but fast scheduling policies are needed to come up with good assignments for assigning tasks to processors.

The example illustrated above is an example of a *machine scheduling problem*, or just *scheduling problem* for short. In scheduling problems a limited number of resources have to be allocated to a number of activities. The goal of this dissertation is to provide for simple but powerful scheduling policies for various scheduling settings. Each chapter deals with a different scheduling problem and studies the quality of a corresponding scheduling policy. The quality of a policy, or actually an algorithm, is measured by comparing the solution which an algorithm might return on some given input to the best solution possible for this input.

In this first chapter I provide an introduction to combinatorial optimization and machine scheduling problems in particular. Since many scheduling problems require too many computations to be solved to optimality for practical purposes, people often look for an algorithm which returns a good solution within a reasonable amount of time. I introduce

simple scheduling policies like list scheduling and local search. Further, I will elaborate on what I understand under 'a good solution' and 'reasonable time' by introducing concepts like polynomial running time, approximation algorithms and notions from complexity theory.

1.1 Combinatorial optimization

In *combinatorial optimization* one is asked to find the best possible solution in terms of some objective function among a large set of alternatives. A combinatorial optimization problem is specified by a set of problem instances. A problem instance, in this case, only allows for either finitely many solutions, or countably infinitely many solutions. A combinatorial optimization is typically either a *minimization* or *maximization* problem, depending on the objective.

Definition 1.1.1. *An instance $\mathcal{I}$ of a combinatorial optimization problem P consists of a finite (or countably infinite) set of feasible solutions S, and an objective function $f : S \to \mathbb{R}$ which assigns a real value to each solution. The problem is to find an* optimal solution*, i. e., a solution $s^* \in S$ such that*

$$f(s^*) \leq f(s) \quad \textit{for all} \quad s \in S$$

in case that P is a minimization problem*, or*

$$f(s^*) \geq f(s) \quad \textit{for all} \quad s \in S$$

in case that P is a maximization problem*.*

The combinatorial optimization problems which are considered in this work are all minimization problems. Thus, from now on I assume optimization problems to be minimization problems and refer to them as just being *problems*. An *instance* of a problem is the actual information for which an optimal solution can be derived whereas a problem is a set of possible instances. Often the problem is described implicitly and in some abstract way. For example, 'Partition the set A of n integers into two subsets A_1 and A_2 such that difference over the sum of the numbers between the two subsets is minimized' is the description of a problem. A particular instance $\mathcal{I}$ for this problem might be 'Partition the set $\{2, 3, 4, 5, 6\}$ into two subsets such that the difference in the sums of both subsets is minimized'. An optimal solution to the preceding problem instance is $A_1 = \{2, 3, 5\}$ and $A_2 = \{4, 6\}$ such that for both A_1 and A_2 the sum of elements equals 10 and hence the difference between both subsets equals zero.

In this work, I define an *algorithm* to be a step-by-step procedure for solving a combinatorial problem which either outputs some solution or outputs that it cannot find one. A naive algorithm to solve a combinatorial optimization would be to consider all possible solutions in the solution set. As already mentioned, such a *complete enumeration* strategy is not practical since it takes too much time. Therefore, in combinatorial optimization, the challenge is to develop more sophisticated algorithms for finding optimal, or near-optimal, solutions relatively fast.

1.2 Machine scheduling problems

In this dissertation, I focus on a special class of combinatorial optimization problems, namely machine scheduling problems. In machine scheduling problems, or just scheduling problems

for short, the aim is to allocate a limited number of resources to a number of activities which require processing. One can think of these resources as being *machines* and of the activities as being *jobs* which need to be processed by these machines. In the example starting of this chapter, the machines were called processors and jobs were called (computing) tasks. A *schedule* σ is then an assignment of jobs to time slots on a particular machine telling when and on which machine a job is being processed.

Besides production and computing environments, applications can be found in personnel planning, maintenance scheduling and all other kinds of service industries. Also, due to the wide variability in different scheduling problems, many other combinatorial optimization problems can be modeled as some scheduling problem. In this section I discuss various scheduling problems by introducing various machine and job environments, different objectives, stochastics and realtime scheduling. Additionally, terminology and notation are fixed.

1.2.1 Three-field notation

Graham, Lawler, Lenstra and Rinnooy Kan [62] classify scheduling problems according to three characteristics: the machine environment α, the job characteristics β and the objective γ. Using their notation I can refer to a specific scheduling problem by it characteristics $\alpha|\beta|\gamma$. I continue with introducing the various entries these fields can take for the scheduling problems to bediscussed in this thesis.

Machine environment. The machine scheduling problems considered in this thesis can be categorized into three different machine environments which, in the notation of Graham et al., are denoted by P, Q and R, i. e., $\alpha \in \{P, Q, R\}$. For all three environments, a set J of n jobs have to be processed by a set M of m parallel machines. The simplest environment is that of *identical parallel machines*, denoted by $\alpha = P$, where each job $j \in J$ has a *processing requirement* p_j. Thus, it takes a machine p_j time units to process job j. In the second machine environment, each machine $i \in M$ has a *speed* s_i. The time it takes machine $i \in M$ to process job $j \in J$ is known as the *processing time* of machine i for job j and is denoted by $p_{ij} = p_j/s_i$. One speaks of *(uniform) related parallel machines* and denotes this setting by $\alpha = Q$. The most general setting is when there is no relationship between the processing time of a job j on two different machines. One could say that $p_{ij} = \frac{p_j}{s_{ij}}$, where s_{ij} is the job-related speed of a particular machine i for a job j. This setting is known as *unrelated parallel machines* and denoted by $\alpha = R$. In all three settings no machine can process more than one job at a time and no job can be processed on two machines simultaneously.

In case, the number of machines m is not part of the input but a fixed constant instead, it is common to add the letter m after the machine environment. For example, by $\alpha = Pm$ I indicate the identical parallel machine problem with a fixed number of machines m. Similarly, for $\alpha = Qm$ and $\alpha = Rm$.

Job characteristics. The second field represents the job characteristics which are given to a job in addition to its processing times p_{ij}. In contrast to the first field, this field can take multiple entries at once. I only elaborate on those job characteristics which are considered in this thesis.

- $\mathcal{M}_j$ *(allowability sets).* A job j is only allowed to be scheduled on machines in the set $\mathcal{M}_j \subseteq M$. The set $\mathcal{M}_j$ is known as the *allowability set* of job j. Alternatively, if $i \in \mathcal{M}_j$, I say job j is *allowable* on machine i. Some authors prefer using the notion of eligibility sets instead of allowability sets. Note that introducing allowability sets actually offers a hybrid between related parallel machines and unrelated parallel machines where the processing time p_{ij} of a job is set to infinity in case $i \notin \mathcal{M}_j$ and is just p_j/s_i otherwise. This setting is also known as *restricted* related parallel machines since a job j is restricted to be processed by machines in $\mathcal{M}_j$.
- *pmtn (preemption).* If preemption is allowed, then the processing of a job on a machine may be interrupted and continued at a later point in time, possibly even on a different machine, without incurring any extra cost. In case preemption is not allowed, then a job which has started processing on some machine must be processed until completion on that machine without any interruptions. Preemption is allowed in Chapters 4 and 5 under the restriction that migration between different machine is not allowed.
- r_j *(release date).* The release date r_j defines the earliest point in time at which job j is available for processing. In case the presence of release dates is not explicitly stated it is assumed that $r_j = 0$ for all jobs $j \in J$.
- d_j *(due dates).* The parameter d_j denotes the due date or deadline of job j, i. e., job j should be completed by time d_j. In Chapters 4 and 5, I speak of *relative due dates* denoted by d_j as well. In the latter setting a job j which is released at time r_j should be finished by time $r_j + d_j$. Instead of due dates, I also sometimes speak of deadlines.

Objective. I only consider objectives involving the completion time of a job, which obviously depends on the schedule σ. The objective in Chapters 2 and 3 is to minimize the *makespan*. The makespan of a schedule is the largest completion time among all jobs. In Chapter 6, I strive for minimizing the total completion time over all jobs, known as minimizing the *sum of completion times*. Finally, in Chapters 4 and 5 a proper objective is absent. Those chapters just care about the existence of a *feasible schedule*. Here, a feasible schedule is one wherein each job meets its relative due date.

1.2.2 Notation and terminology

Given an instance $\mathcal{I}$ of a scheduling problem P, a *schedule* tells for each job during which time slot and by which machine it is being processed. For a given instance $\mathcal{I}$, an arbitrary schedule is denoted by $\sigma(\mathcal{I})$ whereas an optimal schedule (with respect to the objective) is denoted by $OPT(\mathcal{I})$. In all upcoming chapters migration is not allowed such that a job is assigned to only a single machine. Then, let $J_i^{\sigma(\mathcal{I})} \subseteq J$ denote the set of jobs assigned to machine i by schedule $\sigma(\mathcal{I})$. The *processing requirement on a machine* $i \in M$ is defined as $\sum_{j \in J_i^{\sigma(\mathcal{I})}} p_j$ and the *load of a machine* i is defined by $L_i^{\sigma(\mathcal{I})} = \sum_{j \in J_i(\sigma)} p_{ij}$. Thus, in absence of release dates, the load of a machine specifies the moment in time at which the machine completes all the processing requirement which was assigned to it. The completion time of a job j depends on the schedule and hence is denoted by $C_j^{\sigma(\mathcal{I})}$. The *makespan* of a schedule is the moment in time at which the last job completes. This moment coincides with the moment at which the

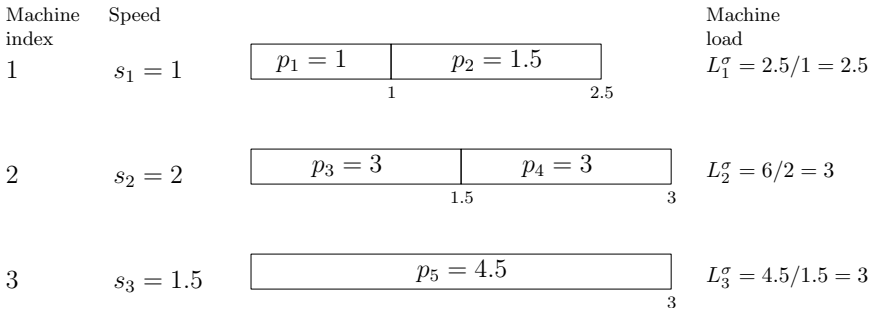

Figure 1.1: A schedule σ for assigning 5 jobs to 3 related parallel machines. Schedule σ has makespan $C_{\max}^{\sigma} = 3$.

last machines completes all processing requirement assigned to it. Hence, the makespan of a schedule $\sigma(\mathcal{I})$ is computed as $C_{\max}^{\sigma(\mathcal{I})} = \max_{j \in J} C_j^{\sigma(\mathcal{I})} = \max_{i \in M} L_i^{\sigma(\mathcal{I})}$ (where the last equation holds in the absence of release dates). A machine whose load equals the makespan is called a *critical machine*.

If the instance $\mathcal{I}$ is clear from the context, I drop the dependence of σ upon $\mathcal{I}$. That is, I write J_i^{σ} instead of $J_i^{\sigma(\mathcal{I})}$, L_i^{σ} instead of $L_i^{\sigma(\mathcal{I})}$, C_j^{σ} instead of $C_j^{\sigma(\mathcal{I})}$, and $C_{\max}^{\sigma}$ instead of $C_{\max}^{\sigma(\mathcal{I})}$. If the schedule σ is clear as well, I simplify our notation further to J_i, L_i, C_j and $C_{\max}$.

The maximum speed and the minimum speed present in an instance $\mathcal{I}$ are denoted by $s_{\max} = \max_{i \in M} s_i$ and $s_{\min} = \min_{i \in M} s_i$ respectively. Similarly, I write $p_{\max} = \max_{j \in J} p_j$ and $p_{\min} = \min_{j \in J} p_j$ for the largest and smallest processing requirement present in an instance. Further, let $S = \sum_{i \in M} s_i$ and $P = \sum_{j \in J} p_j$.

I represent a schedule by a *Gantt chart* where jobs are represented by rectangles which block the processing capability of a machine during some time slot from processing other jobs [52]. In such a graphical representation the vertical axis presents the various machines and the horizontal axis represents time. A schedule σ for scheduling 5 jobs on 3 related parallel machines is presented in Figure 1.1. From the given representation one can also derive the completion time of each job, i. e., in Figure 1.1 job 4 completes on machine 2 at time 3.

1.2.3 Further generalizations

Stochastic scheduling. In traditional scheduling problems, the processing requirements of jobs are deterministic. In *stochastic scheduling* the processing requirement of a job is a random variable. Usually, it is assumed that these random variables are independent. Machine speeds are still assumed to be deterministic. The solution to a stochastic scheduling problem is not merely a simple schedule but a *scheduling policy* instead. Since one does not know the exact processing requirements of jobs, one also does not know when a job is finished and when the next job can start processing on a machine. Thus, decisions need to be taken throughout time. The notion of a scheduling policy as introduced by Möhring, Radermacher, and Weiss [91, 92] states: a scheduling policy specifies actions at decision times t. An action specifies the set of jobs which are processed from time t onwards until the next action is taken

at the next decision point $t' > t$. Usually a new action is taken whenever a job has just been completed but when preemption is allowed this might be at an arbitrary point in time. To determine its next action, a scheduling policy may utilize all the available information which might encompasses knowledge on the existence of all jobs and their release times. However, it may not anticipate on information to be revealed in the future such as the actual realization of the processing requirement of a job which has not yet been completed. That is, a scheduling policy must be *non-anticipatory*.

In Chapter 6, I introduce different classes of jobs such that the processing requirements of all jobs within a class are random variables all having the same probability distribution. The scheduling policies studied are non-anticipatory according to the above definition as they only utilize information which has been revealed. However, some policies do anticipate on the fact that more information about a job class will be revealed in the future when processing a job of this class. Those policies however do not know how that information is going to look like. Therefore, I will still call these policies non-anticipatory as they do not utilize the actual realizations of the processing requirements of still to be processed jobs.

Online time scheduling. In *online time scheduling* jobs arrive over time. The existence of a job and its characteristics are only revealed to a scheduler once the job has arrived. Also here, the solution to an online scheduling problem is represented by a non-anticipatory scheduling policy. Decision points are now often the moments in time at which a job is being completed or a new job arrives. Chapters 4 and 5 present online policies for sporadic task system where a task releases a sequence of jobs throughout time.

Task systems. A generalization of the notion of a job is proposed by the concept of a *task*. A task τ is characterized by three parameters (c_τ, p_τ, d_τ) and releases a sequence of jobs throughout time. All jobs belonging to the same task have the same *processing requirement* c_τ. Further, the points in time at which jobs of the same task are released are at least some minimum separation time p_τ apart, also known as the *period*. Finally, a job should be completed before its *relative deadline* d_τ, that is, a job being released at time t should be finished by time $t + d_\tau$. A *sporadic task system* then consists of a set of such tasks. The case wherein $d_\tau \leq p_\tau$ for all tasks τ is known as *constrained deadlines* whereas the case wherein $d_\tau = p_\tau$ for all tasks τ is known as *implicit deadlines*. Note I stick here to the generally accepted notation within realtime scheduling. That is, the processing time of a job is denoted by c_τ (as opposed to p_j in traditional scheduling) whereas p_τ now denotes a task's period.

1.3 Some well known scheduling algorithms

As some of the theoretically best known algorithms are rather involved and often not suited for practical implementation, I believe in adopting simple heuristics which return good solutions fast. Such powerful but simple scheduling policies are needed to deal with the ever increasing complexity of the optimization problems which practitioners face nowadays. The dissertation laying in front of you adds to this development by studying the quality of some *simple* greedy and local search algorithms for various scheduling problems of the preceding section. While greedy algorithms make reasonable ad hoc decisions to obtain a schedule, local

search algorithms start with some schedule and iteratively improve the current schedule by performing local operations on the current solution until no such improvement can be found anymore. In this work, I study list scheduling algorithms and a local search algorithm with two different neighborhoods.

1.3.1 List scheduling

List scheduling is a greedy algorithm which starts from an empty schedule and a list of jobs. It then repeatedly selects the next unscheduled job from the list and assigns it to the machine where the increase in the objective value is minimum. To provide an example: on an ordered list with nondecreasing processing requirements, in case one strives for minimizing the sum of completion times, the next unscheduled job from the list will be assigned to the machine on which it will be completed the earliest with respect to the current partial schedule.

A famous list scheduling algorithm is *shortest processing time first (SPT)* which orders jobs in the list to their processing requirements. This algorithm has also a variant in stochastic scheduling. The *shortest expected processing time first (SEPT)* orders jobs in the list to their expected processing requirements.

An online version of the list scheduling algorithm is popular for scheduling problems in online time. Call a job *pending* if it has already been released and so is available for processing, but did not yet commence processing. The *earliest deadline first (EDF)* algorithm then will always process the job with the earliest deadline among all pending jobs. If preemption is allowed it might even be that the processing of a job is preempted in case a new job with a tighter deadline arrives, e. g., see Chapters 4 and 5.

1.3.2 Local search

The greedy list scheduling policies mentioned in the preceding subsection only construct a single initial schedule. A follow up procedure might be to try to improve on this initial schedule. Local search provides a framework for such improvement procedures. *Local search* methods iteratively search through the set of feasible solutions. Starting from an initial solution, a local search procedure moves from a feasible solution to a neighboring solution until some stopping criteria are met. A *neighborhood function* defines for each feasible solution a set of solutions which can be generated from the original solution by applying some local operations. To give an example, such a local operation might simply be moving a job to a different machine. The set of neighboring solutions is called the *neighborhood*. The choice of a suitable neighborhood function is important to the performance of a local search algorithm. The simplest form of local search is *iterative improvement*. This method iteratively chooses a better solution in the neighborhood of the current solution and it terminates when no such better solution can be found. The final solution returned will be called a *local optimum* since no better solution exists in the corresponding neighborhood. Note that in general there is no guarantee for a local optimum to be globally optimal.

The local search algorithms studied in this thesis consider neighborhoods which move a single job to a different machine. Moving a job from one machine to another will be called a *jump*. I consider two variations of this jump neighborhood. For the first neighborhood I require that such a jump either reduces the makespan of the schedule, or reduces the number

of critical machines without increasing the makespan. I will refer to this neighborhood as the *jump neighborhood*. The second neighborhood, called the *lexjump neighborhood*, is somewhat more contrived as it requires the updated load of the machine to which the job has moved to be lower than the load of the machine where the job moved away from, i. e., the sorted vector of machines loads needs to decrease lexicographically. Hence, the terminology.

1.4 Computational complexity

Not all scheduling problems are equally hard. Some problems can be solved to optimality fast whereas others cannot be solved to optimality within any reasonable amount of time, even if the size of the input is comparable. It is for those hard problems that I would like to develop simple, good and fast algorithms. To find out how hard a problem actually is, researchers are interested in the amount of computational effort required to solve a certain problem to optimality. Consequently, the current section elaborates on when an algorithm is considered to be *fast*.

Let me start by introducing some terminology. A *decision problem* is a problem where given an input instance $\mathcal{I}$, the output is restricted to be either 'yes' or 'no'. An example of such a decision problem would be the following question: 'Does the scheduling instance provided in Figure 1.1 allow for a schedule with makespan at most 2.5?'. In general one could ask: 'Consider an instance of related parallel machine scheduling. Does there exist a schedule which has makespan of value at most z?'. Note that any minimization (maximization) problem can be translated to a decision problem by asking whether a solution exists having at most (at least) a certain value. A problem is said to be easy if there exists a *polynomial time algorithm* which solves the corresponding decision problem. A polynomial time algorithm is an algorithm which runs in polynomial time in the size of the input, that is, there exists a polynomial q such that the algorithm finds a solution to the input $\mathcal{I}$ in time $q(|\mathcal{I}|)$, where the input size $|\mathcal{I}|$ represents roughly the number of bits needed to represent the instance $\mathcal{I}$. Here, *(running) time* roughly refers to the number of elementary operations performed by the algorithm. Decision problems for which a polynomial time algorithm exists which returns the correct answer, fall within the problem class $\mathcal{P}$. In this thesis however, I focus on a different class of problems, namely problems which are $\mathcal{NP}$*-hard*. For those problems it is not known, and rather unlikely, that an optimal polynomial time algorithm exists.

I will discuss the latter notion more explicitly. Cook [35] formalized the class $\mathcal{NP}$ as the set of decision problems for which each 'yes'-input has a certificate x, such that the size of the certificate $|x|$ is a polynomial in the size of the input $|\mathcal{I}|$, and for which one can verify in polynomial time that x is a valid certificate for $\mathcal{I}$. A *certificate* to the decision problem stated in the previous paragraph would be an actual schedule which indeed has a makespan of at most value z. Clearly, $\mathcal{P}$ is a subset of $\mathcal{NP}$. It is a long standing open question whether $\mathcal{P} = \mathcal{NP}$ holds true. The complexity class *co-*$\mathcal{NP}$ is given by the set of problems whose complement is in $\mathcal{NP}$. In other words, the class *co-*$\mathcal{NP}$ is the set of decision problems for which each 'no'-input has a certificate x, such that the size of the certificate $|x|$ is a polynomial in the size of the input $|\mathcal{I}|$, and for which one can verify in polynomial time that x is a valid certificate for $\mathcal{I}$.

To properly explain the concept of $\mathcal{NP}$*-hardness* and *co-*$\mathcal{NP}$*-hardness*, I need one more

notion. A problem $P_1 \in \mathcal{NP}$ *polynomially reduces* to another problem $P_2 \in \mathcal{NP}$ if there exists a function $\mathcal{F}$ which maps any instance $\mathcal{I}_1$ of P_1 to an instance $\mathcal{I}_2$ of P_2 such that the mapping $\mathcal{F}$ can be computed in polynomial time with respect to the input size of $\mathcal{I}_1$, and such that $\mathcal{I}_2$ is a 'yes'-instance of P_2 if and only if $\mathcal{I}_1$ is a 'yes'-instance of P_1. A decision problem $P_1 \in \mathcal{NP}$ is said to be *$\mathcal{NP}$-complete* if the problem is at least as hard as any other problem in $\mathcal{NP}$, that is, all other problems in $\mathcal{NP}$ polynomially reduce to P_1. Hence, to show some problem P_1 is *$\mathcal{NP}$-complete*, one needs to show that $P_1 \in \mathcal{NP}$ and additionally that there is another *$\mathcal{NP}$-complete* problem P_2 which polynomially reduces to P_1. A combinatorial optimization problem is said to be *$\mathcal{NP}$-hard* if its decision version is *$\mathcal{NP}$-complete*. Similarly, a decision problem $P_1 \in$ *co-$\mathcal{NP}$* is said to be *co-$\mathcal{NP}$-complete* if the problem is at least as hard as any other problem in *co-$\mathcal{NP}$*. A combinatorial optimization problem is said to be *co-$\mathcal{NP}$-hard* if its decision version is *co-$\mathcal{NP}$-complete*.

To conclude the discussion about computational complexity I elaborate on the meaning of the abbreviation $\mathcal{NP}$: Nondeterministically Polynomial time. Problems which are *$\mathcal{NP}$-hard* are at least as hard as any problem in $\mathcal{NP}$. Hence, the belief that $\mathcal{P} \neq \mathcal{NP}$ would imply that no polynomial time algorithm would exist for problems which are *$\mathcal{NP}$-hard*. For many *$\mathcal{NP}$-hard* problems optimal pseudo-polynomial algorithms have been developed though. A *pseudo-polynomial time algorithm* runs in time which is a polynomial in the numeric value of the input (which can be exponential in the size of the input). Alternatively, scientists develop optimal *quasi-polynomial time algorithms* for *$\mathcal{NP}$-hard* problems. A quasi-polynomial time algorithm has a running time which is bounded by $O(2^{(\log q(|\mathcal{I}|))^c})$ for some fixed constant c and a polynomial q depending on the input size $|\mathcal{I}|$ of the problem considered. Thus, a quasi-polynomial time algorithm has a super-polynomial but sub-exponential running time.

1.5 Approximation algorithms

Solving *$\mathcal{NP}$-hard* problems to optimality is unlikely to take only polynomial time and at least requires a vast amount of computational effort. One way to deal with *$\mathcal{NP}$-hard* problems is to look for approximate solutions. An *ρ-approximation algorithm* for a minization problem is guaranteed to deliver a solution which has a value at most ρ times the optimal solution value.

Definition 1.5.1. *An algorithm $\mathcal{A}$ is a ρ-approximation algorithm for a minimization problem P if for any instance $\mathcal{I}$ of P, $\mathcal{A}$ returns a solution which has a value at most ρ times the value of the optimal solution on $\mathcal{I}$. The value $\rho \geq 1$ is known as the* performance guarantee*.*

In this thesis, with a *good* or *powerful* algorithm I imply an algorithm that has a (relatively) low performance guarantee. Some *$\mathcal{NP}$-hard* problems allow for very good approximation algorithms. A family of algorithms $\{\mathcal{A}_\epsilon\}_{\epsilon>0}$ is called a *polynomial time approximation scheme (PTAS)* for some minimization problem P if $\mathcal{A}_\epsilon$ is a polynomial time $(1+\epsilon)$-approximation algorithm, for every fixed $\epsilon > 0$. Similarly, a *quasi-polynomial time approximation scheme (QPTAS)* is a quasi-polynomial time $(1+\epsilon)$-approximation algorithm, for every fixed $\epsilon > 0$. Further, a *fully polynomial time approximation scheme (FPTAS)* is a PTAS $\mathcal{A}_\epsilon$ whose running time is polynomial in not only the input size $|\mathcal{I}|$, but polynomial in $1/\epsilon$ as well.

The performance guarantee is a worst case bound for the performance of an algorithm. On many instances the algorithm might actually perform much better or even optimal. Instances

for which an algorithm actually performs as bad as its performance guarantee are often artificial instances which are rarely encountered in practical applications. Hence, to bring theory closer to practice, people also look at *average case analysis*. Here, one considers the ratio of the value of the solution returned by algorithm over the value of the optimal solution for a given instance, and one averages this ratio over a large range of instances. The drawback of this approach is that it is hard to find the right distribution of instances which match the instances arising in practice. Secondly, say one finds the right distribution of instances, it is probably even harder to perform average case analysis and retrieve relevant performance measures for such real life instances' distributions. Therefore, researchers often settle for an easy distribution like the uniform distribution. The question remaining is how meaningful such results are since often average case analysis is highly sensitive to the distribution of instances selected by the researcher.

1.6 Outline

In this dissertation I study the quality of simple but powerful scheduling policies for various problem settings. The outline of the dissertation is as follows.

The next two chapters study the quality of the jump and the lexjump neighborhood local search algorithms. The objective is to minimize the makespan for restricted related parallel machines. Chapter 2 studies the performance guarantees of these two local search algorithms. Though local search algorithms are often applied in practice as they yield good practical performance, this chapter shows that their worst case analysis might be disappointing. Chapter 3 seeks to bring theory closer to practice. The theoretical performance guarantees are often determined by some rare artificial instances. These instances are fragile in a sense that if the input date is changed slightly, it might be that suddenly the structure of the instance breaks down and the algorithm actually performs quite well on this slightly perturbed instance. In industry, one would not expect such behavior, i. e., if a practical instance is bad, one would expect that the slightly perturbed instance still is bad. Chapter 3 shows that the performance guarantees for both the local search algorithms as well as for the list scheduling algorithms are not robust against these small perturbations. To be more specific, I show that by adding some noise to the input date, the expected performance guarantees of these algorithms drops significantly.

Chapters 4 and 5 treat the question of whether a sporadic task system can feasibly be scheduled on a set of identical or unrelated parallel machines respectively. For the case of unrelated parallel machines, a $8 + 6\sqrt{2/3}$-approximation test is presented in Chapter 4. The algorithm first models the scheduling problem as an integer linear program and applies iterative rounding afterwards. Further, it is shown that no polynomial time algorithm can have a performance guarantee strictly less than 2 unless $\mathcal{P} = \mathcal{NP}$. Further, a PTAS is provided in case the number of machines is constant. In the subsequent chapter, a QPTAS is given for identical parallel machines in case the ratio of the largest relative deadline over the smallest relative deadline is polynomially bounded.

In the final chapter, I study a stochastic scheduling problem in which m classes of independent jobs have to be processed non-preemptively by a single machine. The processing times of the jobs are assumed to be exponentially distributed with parameters depending on the

class of each job. I adopt a Bayesian framework in which the parameters of the job classes are assumed to be unknown. However, by processing jobs from each class, the scheduler can gradually learn about the value of these parameters, thereby enhancing the decision making in the future. I study the qualitative performance of SEPT and an adaptive (dynamic) version of SEPT which I both compare to the quality of an impractical but optimal dynamic program.

1.7 List of publications

This dissertation is based upon the following publications:

Publications:

- Local search performance guarantees for restricted related parallel machine scheduling, with Diego Recalde, Tjark Vredeveld and Petra Schuurman. Appeared in the proceedings of Latin American Theoretical Informatics Symposium 2010 [98]. A journal version appeared in Operations Research Letters 2012 [103].
- Smoothed performance guarantees for local search, with Tobias Brunsch, Heiko Röglin and Tjark Vredeveld. Appeared in the proceedings of European Symposium on Algorithms 2011 [24]. A journal version has been submitted to Mathematical Programming and has been tentavily accepted.
- Learning in stochastic machine scheduling, with Sebastián Marbán and Tjark Vredeveld. Appeared in the proceedings of Workshop of Approximation and Online Algorithms 2011 [86].
- Assigning sporadic tasks to unrelated parallel machines, with Alberto Marchetti-Spaccamela, Suzanne van der Ster and Andreas Wiese. Appeared in the proceedings of International Colloquium on Algorithms and Linear Programming 2012 [87].

Research Memoranda:

- Asymptotic optimality of SEPT in Bayesian scheduling, with Sebastián Marbán and Tjark Vredeveld, Meteor Research Memoranda RM/10/050, Maastricht University.
- Tight performance in Bayesian Scheduling, with Sebastián Marbán and Tjark Vredeveld, Meteor Research Memoranda RM/10/052, Maastricht University.

Submissions:

- Approximating realtime scheduling on identical machines, with Nikhil Bansal, Suzanne van der Ster, Tjark Vredeveld and Ruben van der Zwaan, to be submitted to the Real Time Scheduling Symposium 2013.

Chapter 2

Local search algorithms in scheduling

Local search methods are often applied in practice as they are easy to use and usually yield good solutions fast. The theoretical understanding of those methods though, is unfortunately often limited. Therefore, this chapter studies the performance of two simple but popular jump neighborhoods on the classical scheduling problem of minimizing the makespan on restricted related parallel machines. In particular, I study the performance guarantee of local optima with respect to the jump and the lexicographical jump neighborhood. Additionally, I will remark on two more neighborhoods.

This chapter is based on [98] which is a collaborative work with Diego Recalde, Tjark Vredeveld and Petra Schuurman.

2.1 Introduction

Problem Definition. I study the performance guarantees of two local search algorithms on the problem of minimizing the makespan on restricted related parallel machines. Restricted related parallel machines scheduling is also known as the related parallel machine scheduling with eligibility constraints [80, 81] or as the restricted assignment model for related parallel links [7, 49, 50]. It has applications in, among others, operating systems, communication networks [70], semiconductor manufacturing [27], and throughput management of hospital operating rooms [115]. Adapting the standard notation introduced by [62], the problem of minimizing the makespan on restricted related parallel machines is denoted by $Q|\mathcal{M}_j|C_{\max}$. Since related parallel machine scheduling is already strongly $\mathcal{NP}$*-hard* [54], restricted related machine scheduling is strongly $\mathcal{NP}$*-hard* as well. I refer the reader to [80, 94] and to the survey [79] for an overview of approximation algorithms for restricted related parallel machines.

In restricted related parallel machine scheduling a job $j \in J$ is only allowed to be scheduled on one of its *eligible machines*, $\mathcal{M}_j \subseteq M$. I also say that job j is *allowable* on machine i if $i \in \mathcal{M}_j$. The time it takes machine i to process job j is given by $p_{ij} = p_j/s_i$ if $i \in \mathcal{M}_j$, and $p_{ij} = \infty$ otherwise. Here, p_j denotes the processing requirement of job $j \in J$ and s_i denotes the speed of machine $i \in M$. The objective is to schedule each job to a machine in order

to minimize the makespan, that is, to minimize the moment in time at which the last job completes.

The focus of this chapter will be on the jump and the lexjump neighborhoods. Two more neighborhoods are remarked upon in the final section of this chapter. Local search algorithms and in particular, the jump and lexjump neighborhood, were already briefly introduced in Subsection 1.3.2. The *jump neighborhood* consists of those jumps for which a job j is moved from a *critical* machine i to another machine i' such that $L_{i'}^{\sigma(\mathcal{I})} + p_{i'j} < L_i^{\sigma(\mathcal{I})} = C_{\max}^{\sigma(\mathcal{I})}$ for the given schedule $\sigma(\mathcal{I})$. The *lexjump neighborhood* consists of those jumps which move a job j from *any* machine i to another machine $i' \in \mathcal{M}_j$ such that $L_{i'}^{\sigma(\mathcal{I})} + p_{i'j} < L_i^{\sigma(\mathcal{I})}$ for the given schedule $\sigma(\mathcal{I})$. Note that a lexjump lexicographically decreases the vector of sorted machine loads and hence the terminology. A schedule is *jump (lexjump) optimal* if its jump (lexjump) neighborhood is empty. The notion of a lexjump optimal schedule actually corresponds to the notion of a pure Nash equilibrium in the context of load balancing games when jobs are interpreted as selfish users who would like to be scheduled on a machine of minimal load, e. g., see [116]. Moreover, the price of anarchy of a game is then equivalent to the performance guarantee of a lexjump optimal solution.

Related work. The use of local search in combinatorial optimization problems dates back to 1958 when Bock [20] and Croes [37] introduced the first edge-exchange algorithms for the Traveling Salesman Problem. In the decades to follow, local search algorithms have been developed for a wide range of combinatorial optimization problems due to its simplicity and usually good empirical performance which makes it a powerful tool not only for academics but for many practitioners as well. Aarts and Lenstra [1] present an overview of both theoretical and practical aspects of local search. Other excellent books on local search are [90, 68, 60].

The performance of local search methods within the field of scheduling has been studied before. In (unrestricted) parallel machine scheduling, that is where $\mathcal{M}_j = M$ for all jobs $j \in J$, Finn and Horowitz [47] show that, for identical parallel machines, $2 - 2/(2m+1)$ is an upper bound on the performance guarantee for jump optimal schedules. It follows from Cho and Sahni [33] that $1/2 + \sqrt{m - 3/4}$ is an upper bound on the performance guarantee for related parallel machines. Schuurman and Vredeveld [109] provide instances for which both bounds are tight. The performance guarantee for the lexjump neighborhood on identical parallel machines equals the performance guarantee of the jump neighborhood, see [47, 109]. Czumaj and Vöcking [38] show that the performance guarantee on related parallel machines is in $\Theta\left(\min\left\{\log m / \log\log m, s_{\max}\right\}\right)$ where it is assumed that $s_{\min} = 1$. Recently, results for lexjump optimal schedules on restricted parallel machines have been developed. Awerbuch, Azar, Richter and Tsur [7] and Gairing, Lücking, Mavronicolas and Monien [49] independently showed that the performance guarantee of lexjump optimal schedules for the problem of minimizing the makespan on restricted identical parallel machines is $O\left(\log m / \log\log m\right)$. The latter paper also provides an example showing that the bound is tight up to a constant factor. Hoefer and Souza [66] provide an alternative upper bound for the performance guarantee: $1 + m^2 / \sum_{j \in J} p_j$ where $p_{\max} = 1$. For restricted related parallel machines, Gairing, Lücking, Mavronicolas and Monien [49] show that the performance guarantee is bounded from below by $m - 1$ and bounded from above by m. Since the example establishing the lower bound of $m - 1$ is somewhat artificial, Lu and Yu [84] introduced the concept of λ-*goodness instances*

to develop an alternative performance guarantee. An instance is λ-good if and only if every job can use at least one machine which has a speed of no less than $s_{\max}/\lambda$. Lu and Yu show that for λ-good instances, the performance guarantee is in $\Theta\left(\min\left\{\frac{\log \lambda m}{\log\log \lambda m}, m\right\}\right)$. For the special case where all jobs have identical processing requirements Gairing, Lücking, Mavronicolas and Monien [49] show that the performance guarantee of the lexjump neighborhood on restricted related parallel machines is in $\Theta\left(\log n/\log\log n\right)$.

Contribution. Local search algorithms are popular methods in practice as they are relatively simple to understand and implement and since they usually yield good solutions fast. Unfortunately, from a theoretical point of view, the performance of local search is often poorly understood. This chapter adds to the theoretical understanding of local search algorithms by studying performance guarantees for the jump and lexjump neighborhood for various scheduling settings. Moreover, examples are provided showing that most of the derived performance guarantees are tight.

Outline. The following section gives the performance guarantees of the jump neighborhood for restricted related parallel machines. Section 2.3 provides upper bounds on the performance guarantee for the lexjump neighborhood on restricted related parallel machines which are tight up to a constant factor. Section 2.4 considers the special case of restricted related parallel machines when jobs have identical processing requirements. Finally, Section 2.5 concludes by noting that the bounds provided for the jump neighborhood are also applicable for two more local search neighborhoods which have been studied in literature. Also, this final section remarks on the running time of local search procedures for the scheduling problems discussed in this chapter.

2.2 Jump neighborhood on restricted related machines

This section studies the performance guarantee of the jump neighborhood on restricted related parallel machines. As a special case restricted identical parallel machines are considered. Without loss of generality one may set $s_{\min} = 1$. For the jump neighborhood the following result holds true.

Theorem 2.2.1. *The performance guarantee of the jump neighborhood on restricted related parallel machines is upper bounded by* $1/2 + \sqrt{1/4 + (m-1)s_{\max}}$.

Proof. Consider a jump optimal schedule σ having makespan $C^{\sigma}_{\max}$. Assume w.l.o.g. that machine 1 is a critical machine, i. e., $L^{\sigma}_1 = C^{\sigma}_{\max}$. Let $\mathcal{M}^{\sigma}_1$ be the set of machines to which a job, currently assigned to machine 1 in schedule σ, can be moved, i. e., $\mathcal{M}^{\sigma}_1 = \bigcup_{j \in J^{\sigma}_1} \mathcal{M}_j$. Let $x := |\mathcal{M}^{\sigma}_1|$. Consider a machine $i \in \mathcal{M}^{\sigma}_1$ ($i \neq 1$). Then, there exists at least one job $j \in J^{\sigma}_1$ such that $i \in \mathcal{M}_j$. By jump optimality of σ it follows that, $L^{\sigma}_i + p_j/s_i \geq C^{\sigma}_{\max}$. Consequently, $L^{\sigma}_i \geq C^{\sigma}_{\max} - p_{\max}/s_i$ for all $i \in \mathcal{M}^{\sigma}_1 \setminus \{1\}$. Multiplying the last inequality by s_i and accumulating over all machines $i \in \mathcal{M}^{\sigma}_1$ yields,

$$\sum_{i \in \mathcal{M}^{\sigma}_1} s_i L^{\sigma}_i \geq \sum_{i \in \mathcal{M}^{\sigma}_1} s_i C^{\sigma}_{\max} - \sum_{i \in \mathcal{M}^{\sigma}_1 \setminus \{1\}} p_{\max} = \sum_{i \in \mathcal{M}^{\sigma}_1} s_i C^{\sigma}_{\max} - (x-1)p_{\max}.$$

Since in an optimal schedule machine $i \in M$ can process at most a processing requirement of $s_i C_{\max}^{OPT}$, it follows that the machines in $M \backslash \mathcal{M}_1^\sigma$ need to be able to process a total processing requirement of at least

$$\sum_{i \in \mathcal{M}_1^\sigma} s_i \left(C_{\max}^\sigma - C_{\max}^{OPT} \right) - (x-1) p_{\max}.$$

Therefore,

$$\begin{aligned}
(m-x) s_{\max} C_{\max}^{OPT} &\geq \sum_{i \in M \backslash \mathcal{M}_1^\sigma} s_i L_i^{OPT} \\
&\geq \sum_{i \in \mathcal{M}_1^\sigma} s_i \left(C_{\max}^\sigma - C_{\max}^{OPT} \right) - (x-1) p_{\max} \qquad (2.1) \\
&\geq \sum_{i \in \mathcal{M}_1^\sigma} s_i \left(C_{\max}^\sigma - C_{\max}^{OPT} \right) - (x-1) s_{\max} C_{\max}^{OPT},
\end{aligned}$$

since $p_{\max}/s_{\max} \leq C_{\max}^{OPT}$. Then,

$$\frac{C_{\max}^\sigma}{C_{\max}^{OPT}} \leq \frac{(m-x) s_{\max} + \sum_{i \in \mathcal{M}_1^\sigma} s_i + (x-1) s_{\max}}{\sum_{i \in \mathcal{M}_1^\sigma} s_i} = \frac{(m-1) s_{\max}}{\sum_{i \in \mathcal{M}_1^\sigma} s_i} + 1. \qquad (2.2)$$

As in an optimal schedule OPT the jobs in J_1^σ must be assigned to the machines in $\mathcal{M}_1^\sigma$, it follows that have $s_1 C_{\max}^\sigma \leq \sum_{i \in \mathcal{M}_1^\sigma} s_i \, C_{\max}^{OPT}$. Combining the latter observation with (2.2) yields

$$\frac{C_{\max}^\sigma}{C_{\max}^{OPT}} \leq \min \left\{ \frac{(m-1) s_{\max}}{\sum_{i \in \mathcal{M}_1^\sigma} s_i} + 1 \, ; \sum_{i \in \mathcal{M}_1^\sigma} \frac{s_i}{s_1} \right\} \leq \frac{1}{2} + \sqrt{\frac{1}{4} + (m-1) s_{\max}}.$$

□

By setting $s_i = 1$ for all machines $i \in M$, it follows that any jump optimal schedule on restricted identical machines has makespan at most $1/2 + \sqrt{m - 3/4}$ times the optimal makespan. This bound is tight.

Theorem 2.2.2. *The performance guarantee of the jump neighborhood on restricted identical parallel machines is upper bounded by* $1/2 + \sqrt{m - 3/4}$*, and this bound is tight.*

Proof. The upper bound on the performance guarantee follows straightforward from Theorem 2.2.1 by setting $s_i = 1$ for all machines $i \in M$. The following example shows that the upper bound is tight. Let k be an arbitrary positive integer and consider an instance with $n = k(k-1) + 1$ jobs and $m = n$ machines. All jobs have processing time $p_j = 1$. The first k jobs can only be processed on the first k machines. The remaining jobs are allowable on all machines. Consider the following jump optimal schedule σ. The first k jobs are assigned to machine 1. Machines $2, \ldots, k$ each process $k-1$ of the remaining jobs. This schedule is jump optimal and has a makespan $C_{\max}^\sigma = k$. In an optimal schedule, each machine processes only one job and hence $C_{\max}^{OPT} = 1$. It follows that $C_{\max}^\sigma / C_{\max}^{OPT} = k = 1/2 + \sqrt{m - 3/4}$. □

The tightness example given in Theorem 2.2.2 shows that the bound of Theorem 2.2.1 is tight for $s_{\max} = 1$. The following theorem establishes that the upper bound on the performance guarantee provided in Theorem 2.2.1 is tight up to a constant factor, for arbitrary $m > 2$ and $s_{\max} \geq 2$.

Theorem 2.2.3. *$\sqrt{(m-2)s_{\max}}$ is a lower bound on the performance guarantee for the jump neighborhood on restricted related machines for arbitrary $m > 2$ and $s_{\max} \geq 2$.*

Proof. For any $s_{\max} \geq 2$ and $m > 2$, let $k > 0$ be an integer such that $(k-1)^2 s_{\max} < m - 2 \leq k^2 s_{\max}$. Let the set of machines be partitioned into three subsets M_1, M_2 and M_3. Machine 1 is the only machine in M_1 and has speed 1. The next k machines are in M_2 and have speed $\sqrt{(m-2)s_{\max}}/k$. Finally, all remaining $m-k-1$ machines are in M_3 and have speed $s_{\max}$. Note that $m > (k-1)^2 s_{\max} + 2 \geq k^2 - 2k + 3 \geq k + 1$ for all nonnegative integers k, and consequently at least one machine is in M_3. It is left to validate that the speed of the machines in M_2 is at least the minimum speed (which equals one) and at most the maximum speed. First I show the latter: $\sqrt{(m-2)s_{\max}}/k \leq \sqrt{k^2 s_{\max}^2}/k = s_{\max}$. To show at least a speed of 1, there are two cases. In case $k \geq 2$, $\sqrt{(m-2)s_{\max}}/k > (k-1)s_{\max}/k \geq 2(k-1)/k \geq 1$. Finally, if $k = 1$ then $\sqrt{(m-2)s_{\max}}/k = \sqrt{(m-2)s_{\max}} \geq \sqrt{2} > 1$. The set of jobs consists of two class, i. e., $J = J_1 \cup J_2$. Class J_1 contains of k jobs having processing requirements $p_j = \sqrt{(m-2)s_{\max}}/k$ and allowability sets $\mathcal{M}_j = M_1 \cup M_2$, whereas class J_2 contains $k(m-k-1)$ jobs having processing requirements $p_j = \left((m-2)s_{\max} - \sqrt{(m-2)s_{\max}}\right)/(mk - k^2 - k)$ and allowability sets $\mathcal{M}_j = M$.

In an optimal schedule OPT, the single machine in M_1 remains empty, the machines in M_2 each process one job of class J_1 and all remaining machines each process k jobs of class J_2. It follows that $C_{\max}^{OPT} \leq 1$ since $L_1^{OPT} = 0$, $L_i^{OPT} = 1$ for all machines $i \in M_2$, and

$$L_i^{OPT} = \frac{k * p_j}{s_i} = \frac{(m-2)s_{\max} - \sqrt{(m-2)s_{\max}}}{(m-k-1)s_{\max}} < \frac{(m-2)s_{\max} - \sqrt{(k-1)^2 s_{\max}^2}}{(m-k-1)s_{\max}} = 1,$$

for all machines $i \in M_3$. The following schedule σ is jump optimal: schedule all jobs of J_1 to machine 1, and each machine in M_2 processes $m-k-1$ jobs of J_2. All machines in M_3 remain empty. The jump optimal schedule is also illustrated in Figure 2.1. It follows that $L_1^\sigma = (k\sqrt{(m-2)s_{\max}}/k)/1 = \sqrt{(m-2)s_{\max}}$, and

$$L_i^\sigma = (m-k-1)\frac{p_j}{s_i} = \frac{\left((m-2)s_{\max} - \sqrt{(m-2)s_{\max}}\right)}{k s_{\max}} \leq \sqrt{(m-2)s_{\max}} - 1 < L_1^\sigma,$$

for all machines $i \in M_2$. Since $L_i^\sigma + p_j/s_i = L_1^\sigma$ for all jobs $j \in J_1^\sigma$ and all machines in $i \in M_2$, schedule σ is jump optimal. □

2.3 Lexjump neighborhood on restricted related machines

For the same setting, I provide in the current section a bound on the performance guarantee of the lexjump neighborhood. The proof of this result makes use of the Gamma function. The Gamma function is denoted by $\Gamma(x)$ and is defined by $\Gamma(x) := \int_0^\infty t^{x-1} e^{-x} \partial t$ for $x \in \mathbb{R}^+$. For any natural number N the gamma function is defined as $\Gamma(N+1) := N!$. Let $\Gamma^{-1}(x)$ denote the inverse Gamma function. It is known that both the Gamma function as well as the inverse Gamma function are monotonically increasing and that $\Gamma^{-1}(x) = \Theta(\log x / \log\log x)$ for $x \in \mathbb{R}^+$. Further, the analysis relies on the following machine classification introduced by [116]. Let $C_{\max}^{OPT}$ denote the optimal makespan (for brevity of notation I disregard the

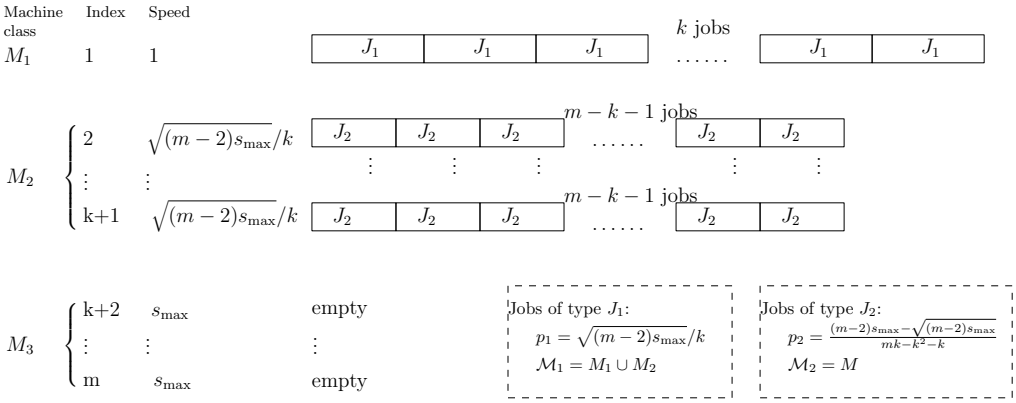

Figure 2.1: The jump optimal schedule σ yielding Theorem 2.2.3.

dependency on the instance $\mathcal{I}$). For a schedule σ and for any non-negative integer k, I define the set $R_k^\sigma = \left\{i \in M : L_i^\sigma \geq k \cdot C_{\max}^{OPT}\right\}$, i. e., the set of machines with load greater or equal than $k \cdot C_{\max}^{OPT}$. Also, let $\overline{R_k^\sigma} = R_k^\sigma \backslash R_{k+1}^\sigma$, i. e., the set of machines with a load at least $k \cdot C_{\max}^{OPT}$ but no more than $(k+1) \cdot C_{\max}^{OPT}$.

Lemma 2.3.1. *Let σ be a lexjump optimal schedule. Then, a job $j \in J_i^\sigma$ scheduled on a machine $i \in R_{k+1}^\sigma$ cannot be assigned to a machine $h \in M \setminus R_k^\sigma$ in any optimal schedule OPT.*

Proof. Let σ be a lexjump optimal schedule. Consider a job $j \in J_i^\sigma$ such that $i \in R_{k+1}^\sigma$. If there is a machine $h \in \mathcal{M}_j \backslash R_k^\sigma$ then

$$k\, C_{\max}^{OPT} + \frac{p_j}{s_h} > L_h^\sigma + \frac{p_j}{s_h} \geq L_i^\sigma \geq (k+1) C_{\max}^{OPT}.$$

The second inequality follows from σ being lexjump optimal. The string of inequalities yields that $p_j/s_h > C_{\max}^{OPT}$, and consequently, in any optimal schedule OPT job j has to be assigned to a machine in R_k^σ. □

Making use of the previous insight, Theorem 2.3.2 follows. Recall that S denotes the sum of all speeds.

Theorem 2.3.2. *The performance guarantee of the lexjump neighborhood for the problem of minimizing the makespan on restricted related parallel machines is in $O(\log S / \log\log S)$.*

Proof. Consider a lexjump optimal schedule σ with makespan $C_{\max}^\sigma$. By definition of R_{k+1}^σ, the processing requirement the machines in R_{k+1}^σ process is at least $(k+1) C_{\max}^{OPT} \sum_{i \in R_{k+1}^\sigma} s_i$. Lemma 2.3.1 yields that any job scheduled by σ to a machine in R_{k+1}^σ needs to be scheduled to a machine in R_k^σ in an optimal schedule OPT. Therefore, in OPT, the machines in $\overline{R_k^\sigma}$ need to process at least a total processing requirement of $k \sum_{i \in R_{k+1}^\sigma} s_i\, C_{\max}^{OPT}$. Hence, for all k, it holds that $\sum_{i \in \overline{R_k^\sigma}} s_i\, C_{\max}^{OPT} \geq \sum_{i \in R_{k+1}^\sigma} k\, s_i\, C_{\max}^{OPT}$, from which it follows that

$$\sum_{i \in R_k^\sigma} s_i = \sum_{i \in \overline{R_k^\sigma}} s_i + \sum_{i \in R_{k+1}^\sigma} s_i \geq k \sum_{i \in R_{k+1}^\sigma} s_i + \sum_{i \in R_{k+1}^\sigma} s_i = (k+1) \sum_{i \in R_{k+1}^\sigma} s_i.$$

Letting $c = \lfloor C^{\sigma}_{\max}/C^{OPT}_{\max} \rfloor$, it follows that $\sum_{i \in R^{\sigma}_0} s_i \geq c! \sum_{i \in R^{\sigma}_c} s_i$. Since all speeds are at least 1 and as the critical machine is in R^{σ}_c it follows that $\sum_{i \in R^{\sigma}_c} s_i \geq |R^{\sigma}_c| \geq 1$. The latter observation and $M = R^{\sigma}_0$ yield $\sum_{i \in M} s_i \geq c! = \Gamma(c+1)$. Using the fact that the inverse of the Gamma function is monotonically increasing, the latter inequality yields, $C^{\sigma}_{\max}/C^{OPT}_{\max} \leq c+1 \leq \Gamma^{-1}(S)$, that is, $C^{\sigma}_{\max}/C^{OPT}_{\max} \in O(\log S/\log\log S)$. □

Note that the bound given in the above theorem matches the bound of Awerbuch et al, [7] in case of restricted identical machines, that is when $s_i = 1$ for all machines $i \in M$. I show next that there exist instances for which the bound of Theorem 2.3.2 is tight up to a constant factor.

Theorem 2.3.3. *The performance guarantee of the lexjump neighborhood for the problem of minimizing the makespan on restricted related parallel machines is in* $\Omega(\log S/\log\log S)$.

Proof. Let $k > 1$ be an arbitrary strictly positive integer and let $s > 1$. Consider the following instance and a corresponding schedule σ. Each job has a processing requirement $p_j = s$. The machines are partitioned into $sk+1$ groups, $S_0, S_1, \ldots, S_{sk}$. Group S_0 consists of only one machine which has a speed of one. For $l = 1, \ldots, sk$, group S_l contains $k\Pi_{i=1}^{l-1}(sk-i)$ machines each having a processing speed of s. In schedule σ, each machine in group S_l, for $l \geq 1$, processes $sk-l$ jobs. Moreover, k jobs are scheduled on the machine in S_0. Each job $j \in J^{\sigma}_i$ with $i \in S_l$ has $\mathcal{M}_j = S_l \cup S_{l+1}$. The schedule is visualized in Figure 2.2. Schedule σ is lexjump optimal with makespan sk, whereas $C^{OPT}_{\max} = 1$. The optimal solution is attained by assigning one job to each machine $i \in S_l$ for $l \geq 1$ and leaving the machine in S_0 empty. Moreover,

$$\sum_{i \in M} s_i = 1 + sk \sum_{i=0}^{sk-1} \frac{(sk-1)!}{i!} < 1 + (sk)! \sum_{i=0}^{+\infty} \frac{1}{i!} = 1 + e\,(sk)! \leq (sk+1)! = \Gamma(sk+2).$$

Using the fact that the inverse of the Gamma function is monotonically increasing, $C^{\sigma}_{\max}/C^{OPT}_{\max} = sk \geq \Gamma^{-1}(S) - 2$. Thus, there exist instances for which

$$C^{\sigma}_{\max}/C^{OPT}_{\max} \in \Omega(\log S/\log\log S).$$

□

Theorem 2.3.2 establishes an upper bound on the performance guarantee of a lexjump optimal schedule. Gairing et al. [49] provided an alternative upper bound of m on the performance guarantee for lexjump optimal schedule for restricted related parallel machines. It can be shown that neither result implies the other result, i. e., one can construct examples for which the bound of Theorem 2.3.2 is tight and the bound provided by Gairing et al. is not, and vice versa.

2.4 Jump neighborhood with identical jobs

Gairing et al. [49] considered the special case of scheduling on restricted related parallel machines where jobs have identical processing requirements, say $p_j = 1$ for all jobs $j \in J$. They showed that the performance guarantee of the lexjump neighborhood is at most $O(\log n/\log\log n)$ in this setting. Further, it is easy to construct an instance and corresponding

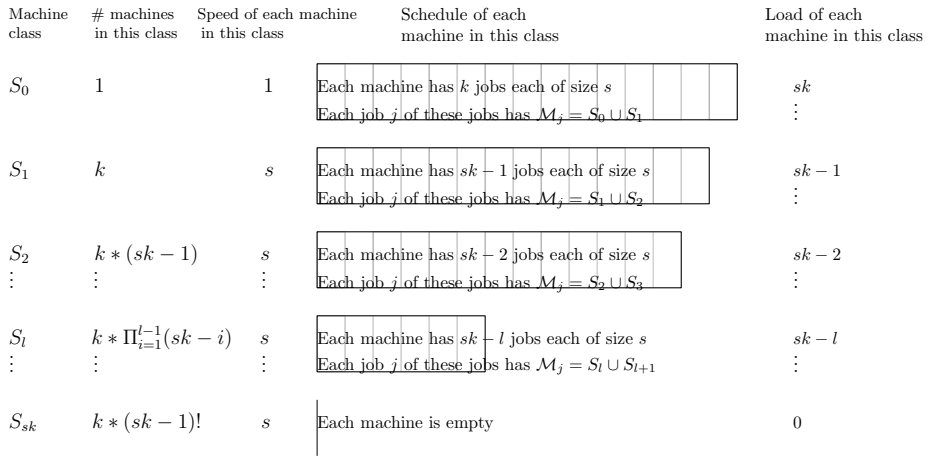

Figure 2.2: The lexjump optimal schedule giving the lower bound in Theorem 2.3.3.

lexjump optimal schedule σ for which $C^{\sigma}_{\max}/C^{OPT}_{\max}$ is in $\Omega(\log n/\log\log n)$. For completeness of the results, this section presents the performance of the jump neighborhood in case of identical jobs and restricted related parallel machines.

Theorem 2.4.1. *The performance guarantee of the jump neighborhood on restricted related parallel machines with identical (unit-length) jobs is upper bounded by* $\sqrt{S+S(m-1)/n}$.

Proof. Assume without loss of generality that machine 1 is a critical machine. Using the same arguments as in the proof of Theorem 2.2.1, the amount of processing requirements assigned to the machines not in $\mathcal{M}^{\sigma}_1$ in the optimal schedule is at least

$$\begin{aligned}\sum_{i\in M\setminus\mathcal{M}^{\sigma}_1} s_i C^{OPT}_{\max} &\geq (1-x) + \sum_{i\in\mathcal{M}^{\sigma}_1}\left(s_i(C^{\sigma}_{\max}-C^{OPT}_{\max})\right)\\ &\geq (1-m)\,\frac{S}{n}C^{OPT}_{\max} + \sum_{i\in\mathcal{M}^{\sigma}_1}\left(s_i(C^{\sigma}_{\max}-C^{OPT}_{\max})\right). \qquad (2.3)\end{aligned}$$

where $x=|\mathcal{M}^{\sigma}_1|$. The first inequality follows from (2.1) and the equation $p_{\max}=1$, whereas second inequality follows from $\sum_{i\in M} s_i C^{OPT}_{\max} \geq \sum_{j\in J} p_j = n$. Since the processing requirement scheduled on machine 1 in the jump optimal schedule σ needs to be processed by the machines in $\mathcal{M}^{\sigma}_1$, it follows that $s_1 C^{\sigma}_{\max} = s_1 L^{\sigma}_1 \leq C^{OPT}_{\max}\sum_{i\in\mathcal{M}^{\sigma}_1} s_i$. Combining the latter observation with (2.3) yields,

$$\frac{C^{\sigma}_{\max}}{C^{OPT}_{\max}} \leq \min\left\{\frac{S+S(m-1)/n}{\sum_{i\in\mathcal{M}^{\sigma}_1} s_i}\,;\,\frac{\sum_{i\in\mathcal{M}^{\sigma}_1} s_i}{s_1}\right\} \leq \sqrt{S+S(m-1)/n}.$$

□

To complete the above result, I show that there exist values for the parameter m, n and S, such that the upper bound on the performance guarantee provided in Theorem 2.4.1 is tight up to an arbitrary small constant.

Theorem 2.4.2. *There exists an instance of minimizing the makespan on restricted related parallel machines with identical (unit-length) jobs such that for any $\epsilon > 0$, there is a jump optimal schedule σ for which*

$$\frac{C^{\sigma}_{\max}}{C^{OPT}_{\max}} > \sqrt{S + S(m-1)/n} - \epsilon.$$

Proof. Let k be an arbitrary strictly positive integer and consider the following instance and a jump optimal schedule σ. Each job has processing time $p_j = 1$ as is required in this section. Then, there are three machines for which $s_1 = s_{\min} = 1$, $s_2 = k-1$ and $s_3 = k(k-1)-1 = s_{\max}$. The set of jobs consists of k jobs which can only be scheduled on the first two machines, and $k(k-1)-1$ which are allowable on machines 2 and 3. Jump optimal schedule σ schedules the first group of jobs on machine 1 and the second group of jobs on machine 2. It follows that $L^{\sigma}_1 = C^{\sigma}_{\max} = k$, $L^{\sigma}_2 = k - 1/(k-1)$ and $L^{\sigma}_3 = 0$. An optimal schedule is obtained by assigning one job to machine 1, $k-1$ jobs to machine 2, and all remaining jobs to machine 3. Consequently, $1 + C^{\sigma}_{\max}/C^{OPT}_{\max} = 1 + k \geq \sqrt{k^2+1}$. Furthermore, $\sqrt{S + S(m-1)/n} = \sqrt{k^2+1}$. By letting k tend to infinity the relative gap between the upper and lower bound becomes arbitrarily small. □

2.5 Concluding remarks

Other neighborhoods. In literature two more neighborhoods have been studied for the problem of minimizing the makespan on related parallel machines. Schuurman and Vredeveld [109] consider the *swap neighborhood* and additionally introduce the *push neighborhood*. A *swap* interchanges the machine assignments of two jobs. The swap neighborhood consists of those swaps and jumps which either reduce the makespan, or decrease the number of machines without increasing the makespan. The *push neighborhood* is a more involved neighborhood which allows for more than one job to be re-asssigned in a single *push* operation. For the precise definition of this neighborhood though, I refer to [109]. Since it can be shown that both swap optimal schedules as well as push optimal schedules are jump optimal, the upper bounds on the performance guarantees provided in Theorems 2.2.1, 2.2.2 and 2.4.1 directly carry over to the swap and the push neighborhood. Moreover, these bounds are also tight for both neighborhoods since the tightness examples provided for the jump neighborhood are all both swap and push optimal.

Running time. This chapter treated the quality of the local optima with respect to four neighborhoods. Another interesting question is how many iterations an iterative improvement procedure needs to find these local optima. Brucker, Hurink and Werner [23], Hurkens and Vredeveld [69] and Schuurman and Vredeveld [109] provide upper bounds on the number of iterations needed to find a jump optimal schedule for the identical parallel machines and the related parallel machines when eligibility constraints are absent. Gairing, Lücking, Mavronicolas, Monien and Spirakis [51] and Feldmann, Gairing, Lücking, Monien and Rode [46] give bounds on the number of iterations needed by an iterative improvement to find a lexjump optimal schedule for identical and related parallel machines, respectively. Finally, on

restricted identical parallel machines, Gairing, Lücking, Mavronicolas and Monien [50] provide a polynomial time algorithm, called 'nashify', which converts any schedule into a lexjump optimal schedule. The algorithm however is not iteratively improving. I conjecture that a procedure exists which reaches a jump or lexjump optimal schedule on restricted related parallel machines in polynomial time (but not necessarily iteratively improving).

Chapter 3

Smoothed analysis for machine scheduling problems

I started the previous chapter by arguing that local search methods are often applied in practice as they yield good empirical performance. However, from a theoretical point of view the performance guarantee of local search algorithms can on some instances be quite bad with respect to more advanced approximation algorithms. In the last decade the concept of *smoothed analysis* has been developed in an effort to decrease the gap between worst case theoretical performance and the empirical performance on real life data. Informally speaking, smoothed analysis moves away from worst case analysis by adding some noise to a bad instance selected by some adversary. If the bad instances for some algorithm are isolated in the space of instances, then adding some noise to these instances will destroy the structure of the worst case instance, yielding a good performance of the algorithm.

This chapter studies the local search algorithms discussed in Chapter 2 on (restricted) related parallel machines. In particular, I study the smoothed performance guarantees of these algorithms for the given scheduling problems. While the lower bounds for all scheduling variants with restricted machines are rather robust, we will see that the bounds are fragile for unrestricted machines. In particular, the smoothed performance guarantee of the jump and the lexjump algorithm are (in contrast to the worst case) independent of the number of machines. They are $\Theta(\phi)$ and $\Theta(\log \phi)$, respectively, where $1/\phi$ is a parameter measuring the magnitude of the perturbation. The latter immediately implies that also the smoothed price of anarchy is $\Theta(\log \phi)$ for routing games on parallel links. Additionally, I remark that for unrestricted machines also the greedy list scheduling algorithm has an approximation guarantee of $\Theta(\log \phi)$.

This chapter is based on [24] and is collaborative work with Tobias Brunsch, Heiko Röglin and Tjark Vredeveld [1].

3.1 Introduction

Problem Definition. This chapter presents a follow up to the preceding chapter. Hence, I refer to Chapter 2 and also to Section 1.2 for introducing (restricted) related parallel

[1] Tobias Brunsch and I both contributed at least one third of the total work to this project.

machines. Similarly, the reader is referred to Section 1.3 for the concepts of local search, the jump and the lexjump neighborhood and also list scheduling. Here, I just remark that in the current chapter it is assumed without loss of generality that $s_{\min} = 1$ and that $p_{\max} \leq 1$. In the remainder of this section, I restrict to introducing and motivating the concept of smoothed analysis.

The performance guarantee of local search and greedy algorithms for scheduling problems is well studied and understood. For most algorithms, matching upper and lower bounds on their performance guarantee are known. For example, consider the performance guarantees known for the machine environments studied here. The 'worst case' column of Table 3.1 shows that the performance guarantees for the jump and lexjump neighborhood are tight up to a constant factor. The performance guarantees of the jump and lexjump neighborhood are constant only for the simplest case with unrestricted and identical machines. In all other cases it increases with the number m of machines. For list scheduling on unrestricted related parallel machines, Aspnes et al. [5] showed that the performance guarantee of list scheduling is $\Theta(\log m)$, i. e., also increasing in the number of machines.

In general, for most algorithms the matching lower bounds are often somewhat contrived. It is therefore questionable whether those worst case instances resemble typical instances seen in practical applications. For that reason, my co-authors and me started to study algorithms in the framework of smoothed analysis, in which instances are subject to some degree of random noise. The underlying idea is to find out for which heuristics and scheduling variants the lower bounds are robust and for which they are fragile and not very likely to occur in practical applications. This chapter aims to find out whether the bounds on the performance guarantees derived in Chapter 2 are robust against a small degree of such random noise.

Smoothed analysis has been introduced by Spielman and Teng [112] to explain why certain algorithms perform well in practice in spite of a poor worst case running time. Smoothed analysis is a hybrid of average case and worst case analysis: First, an adversary chooses an instance. Second, this instance is slightly randomly perturbed. The smoothed performance is the expected performance, where the expectation is taken over the random perturbation. The adversary, trying to make the algorithm perform as bad as possible, chooses an instance that maximizes this expected performance. This assumption is made to model that often the input an algorithm gets is subject to imprecise measurements, rounding errors, or numerical imprecision. If the smoothed performance of an algorithm is small, then bad worst case instances might exist, but one is very unlikely to encounter them if instances are subject to some small amount of random noise.

This chapter follows the more general model of smoothed analysis introduced by Beier and Vöcking [18]. In this model an adversary is even allowed to specify the probability distribution of the random noise. The influence the adversary can exert is described by a parameter $\phi \geq 1$ denoting the maximum density of the noise. Formally, the following input model is considered: the adversary chooses the number m of machines and, in the case of non-identical machines, arbitrary machine speeds $s_{\max} = s_1 \geq \ldots \geq s_m = s_{\min}$. He also chooses the number n of jobs and, in the case of restricted machines, an arbitrary set $\mathcal{M}_j \subseteq M$ for each job $j \in J$. The only perturbed part of the instance are the processing requirements p_j. I define a *ϕ-smooth* instance Φ in the following way:

	worst case		ϕ-smooth	
	jump	lexjump	jump	lexjump
unrestricted identical	$\Theta(1)$ [47, 109]	$\Theta(1)$ [47, 109]	$\Theta(1)$	$\Theta(1)$
unrestricted related	$\Theta\left(\sqrt{m}\right)$ [33, 109]	$\Theta\left(\frac{\log m}{\log\log m}\right)$ [38]	$\Theta(\phi)$	$\Theta(\log \phi)$
restricted identical	$\Theta\left(\sqrt{m}\right)$ [98]	$\Theta\left(\frac{\log m}{\log\log m}\right)$ [7]	$\Theta\left(\sqrt{m}\right)$	$\Theta\left(\frac{\log m}{\log\log m}\right)$
restricted related	$\Theta\left(\sqrt{m \cdot \frac{s_{\max}}{s_{\min}}}\right)$ [98]	$\Theta\left(\frac{\log S}{\log\log S}\right)$ [98]	$\Theta\left(\sqrt{m \cdot \frac{s_{\max}}{s_{\min}}}\right)$	$\Omega\left(\frac{\log m}{\log\log m}\right)$

Table 3.1: Worst case and smoothed performance guarantees for jump and lexjump optimal schedules. Here, $S = \sum_{i=1}^{m} s_i/s_{\min}$.

Definition 3.1.1. *For each p_j the adversary can choose an arbitrary probability density $f_j : [0,1] \to [0,\phi]$ according to which it is chosen, independently of the processing requirements of the other jobs. An instance of this kind is called* ϕ-smooth *and denoted by* Φ.

Formally, a ϕ-smooth instance Φ is not a single instance but a distribution over instances. An instance resulting from one particular realization of Φ is denoted by $\mathcal{I}$.

The parameter ϕ specifies how close the analysis is to a worst case analysis. The adversary can, for example, choose for every p_j an interval of length $1/\phi$ from which it is drawn uniformly at random. For $\phi = 1$, every processing requirement is uniformly distributed over $[0,1]$, which represents one way of performing average case analysis. When ϕ gets larger, the adversary can specify the processing requirements more and more precisely, and for $\phi \to \infty$ the smoothed analysis approaches a worst case analysis. The *smoothed performance guarantee* of the jump and the lexjump neighborhoods is then defined to be the worst case expected performance guarantee over all ϕ-smooth instances.

Related work. For the bounds on the performance guarantees for the jump and lexjump neighborhood I again refer to Section 2.1 of the preceding chapter. Here, I focus on literature using smoothed analysis. Up to now, smoothed analysis has been mainly applied to running time analysis (see, e.g., [113] for a survey). The first exception is the paper by Becchetti et al. [16] who introduced the concept of smoothed competitive analysis, which is equivalent to smoothed performance guarantees for online algorithms. Schäfer and Sivadasan [106] performed a smoothed competitive analysis for metrical task systems. Englert et al. [44] considered the 2-Opt algorithm for the traveling salesman problem and determined, among others, the smoothed performance guarantee of local optima of the 2-Opt algorithm. Hoefer and Souza [66] presented one of the first average case analysis for the price of anarchy.

Contribution. This chapter contributes to literature by raising attention to the use of smoothed analysis to study the qualitative performance of algorithms and local search algorithms in particular. The results for the jump and lexjump neighborhoods are summarized in the last two columns of Table 3.1. The first remarkable observation is that the smoothed performance guarantees for all variants of restricted machines are robust against random

noise. That is, even for large perturbations with constant ϕ, the worst case lower bounds carry over. This can be seen as an indication that neither the jump neighborhood nor the lexjump neighborhood yield a good qualitative performance for scheduling with restricted machines in practice.

The situation is much more promising for the unrestricted variants. In that case, the worst case bounds are fragile and do not carry over to the smoothed case. The interesting case is the one of unrestricted related machines. Even though both for jump and for lexjump neighborhoods the worst case lower bound is not robust, there is a significant difference between these two: while the smoothed performance guarantee for jump grows linearly with the perturbation parameter ϕ, it grows only logarithmically in ϕ for lexjump optimal schedules. This proves that also in the presence of random noise lexjump optimal schedules are significantly better than jump optimal schedules. Additionally, the smoothed performance guarantee of list scheduling is in $\Theta(\log \phi)$ as well. This indicates that both the lexjump algorithm and the list scheduling algorithm should yield good approximations on practical instances.

Moreover, since a lexjump optimal schedule can be interpret as a pure Nash equilibria, the results also imply that the smoothed price of anarchy is in $\Theta(\log \phi)$, showing that known worst case results are too pessimistic in the presence of some noise.

Outline. The chapter continues with Sections 3.2 and 3.3 which study the smoothed performance guarantee of the jump and lexjump neighborhood. Section 3.2 provides asymptotically matching upper and lower bounds on the smoothed performance guarantees of the jump and lexjump neighborhoods for unrestricted related parallel machines. The section thereafter shows that smoothing does not help for the setting of restricted related parallel machines. Section 3.4 moves away from the area of scheduling and seeks to apply the concept of smoothed analysis to the game theoretic problem of routing games and its performance measure, the price of anarchy.

3.2 Related parallel machines

3.2.1 Jump optimal schedules

Theorem 3.2.2 shows that the smoothed performance guarantee for jump optimal schedules on unrestricted related parallel machines grows linearly with the smoothing parameter ϕ and is independent of the number of jobs and machines. In particular, it is constant if the smoothing parameter is constant. In proving our results I make use of the following proposition which follows from Cho and Sahni [33]. This proposition yields an alternative upper bound on the performance guarantee of the jump neighborhood on related parallel machines (in a traditional, non-smoothed setting). $\mathrm{Jump}(\mathcal{I})$ represents the set of all possible jump optimal schedules for an instance $\mathcal{I}$.

Proposition 3.2.1. *For any scheduling instance $\mathcal{I}$ with m related parallel machines and n jobs*

$$\max_{\sigma(\mathcal{I}) \in \mathrm{Jump}(\mathcal{I})} \frac{C_{\max}^{\sigma(\mathcal{I})}}{C_{\max}^{OPT(\mathcal{I})}} \leq \frac{1 + \sqrt{4 \min\{m,n\} - 3}}{2} \leq \frac{1}{2} + \sqrt{n}\,. \tag{3.1}$$

In the remainder of the chapter I use the notation $\mathcal{I} \sim \Phi$ to indicate that the (deterministic) instance $\mathcal{I}$ represents a realization of the (stochastic) instance Φ.

Theorem 3.2.2. *For any ϕ-smooth instance Φ with related parallel machines,*

$$\mathop{\mathbf{E}}_{\mathcal{I}\sim\Phi}\left[\max_{\sigma(\mathcal{I})\in\mathrm{Jump}(\mathcal{I})} \frac{C_{\max}^{\sigma(\mathcal{I})}}{C_{\max}^{OPT(\mathcal{I})}}\right] < 5.1\phi + 2.5 = O(\phi)\,.$$

Proof. First note that if $m > n$, then there both exist an optimal schedule and a worst jump optimal schedule such that neither of both assign any job on any of the slowest $m-n$ machines. Therefore, it is safe to assume that $m \leq n$.

First, I show an upper bound on the performance guarantee of jump optimal schedules which decreases when the sum of processing requirements P increases and which is valid for every (traditional) instance. The second part of the proof considers the effect of smoothing the processing requirements. It will be shown that for ϕ-smooth instances the total processing requirement P is usually not too small, which yields the desired result.

Consider a scheduling instance $\mathcal{I}$ and a corresponding jump optimal schedule $\sigma(\mathcal{I})$. Let i be an arbitrary machine, let machine i^* be a critical machine in schedule $\sigma(\mathcal{I})$, and let j be a job assigned to machine i^* in schedule $\sigma(\mathcal{I})$. By jump optimality of $\sigma(\mathcal{I})$ it follows that

$$C_{\max}^{\sigma(\mathcal{I})} = L_{i^*}^{\sigma(\mathcal{I})} \leq L_i^{\sigma(\mathcal{I})} + p_j/s_i \leq L_i^{\sigma(\mathcal{I})} + p_{\max}/s_i.$$

The previous inequality yields that $s_i \cdot C_{\max}^{\sigma(\mathcal{I})} \leq s_i \cdot L_i^{\sigma(\mathcal{I})} + p_{\max}$ for all machines $i \in M$. Summing the latter inequality over all machines from $M \setminus \{i^*\}$ and adding $s_{i^*} \cdot L_{i^*}^{\sigma(\mathcal{I})} = s_{i^*} \cdot C_{\max}^{\sigma(\mathcal{I})}$ to both sides of the inequality yields

$$\sum_{i\in M} s_i \cdot C_{\max}^{\sigma(\mathcal{I})} \leq \sum_{i\in M\setminus\{i^*\}} p_{\max} + \sum_{i\in M} s_i \cdot L_i^{\sigma(\mathcal{I})} = (m-1)\cdot p_{\max} + \sum_{i\in M} s_i \cdot L_i^{\sigma(\mathcal{I})}\,.$$

As $\sum_{i\in M} s_i \cdot L_i^{\sigma(\mathcal{I})} = \sum_{j\in J} p_j = P$ the following upper bound on the makespan of any jump optimal schedule $\sigma(\mathcal{I})$ follows:

$$C_{\max}^{\sigma(\mathcal{I})} \leq \frac{P}{\sum_{i\in M} s_i} + \frac{m-1}{\sum_{i\in M} s_i} \leq \frac{P}{\sum_{i\in M} s_i} + \frac{n-1}{\sum_{i\in M} s_i}\,,$$

where inequalities follow from $p_{\max} \leq 1$ and $m \leq n$. Using the well-known bound $C_{\max}^{OPT(\mathcal{I})} \geq P/\sum_{i\in M} s_i$ it follows that

$$C_{\max}^{\sigma(\mathcal{I})} \leq \frac{P}{\sum_{i\in M} s_i} + \frac{n-1}{\sum_{i\in M} s_i} \leq \left(1 + \frac{n-1}{P}\right)\cdot C_{\max}^{OPT(\mathcal{I})}\,.$$

Hence,

$$\max_{\sigma(\mathcal{I})\in\mathrm{Jump}(\mathcal{I})} \frac{C_{\max}^{\sigma(\mathcal{I})}}{C_{\max}^{OPT(\mathcal{I})}} \leq 1 + \frac{n-1}{P}\,. \tag{3.2}$$

In the remainder of the proof, let $\mathcal{I}$ be an instance which presents a realization of the random instance Φ. The performance guarantee of any jump optimal schedule $\sigma(\mathcal{I})$ can only be bad if P is small. As the instance $\mathcal{I}$ is now ϕ-smooth, the processing requirements are random

variables in $[0,1]$ with bounded densities. Let $\mathcal{F}$ denote the failure event that $P \leq (n - \sqrt{n \ln n})/(2\phi)$. Define x_j to be independent random variables drawn uniformly from $[0, 1/\phi]$ for all $j \in J$. Then, $\mathbf{Pr}[p_j \geq a] \geq \mathbf{Pr}[x_j \geq a]$ for any $a \in [0,1]$. Let $X = \sum_{j \in J} x_j$. Then, for any $a \in [0, n]$, it follows that $\mathbf{Pr}[P \leq a] \leq \mathbf{Pr}[X \leq a]$. Also, $\mathbf{E}[X] = n/(2\phi)$. Hence,

$$\begin{aligned}\mathbf{Pr}[\mathcal{F}] &= \mathbf{Pr}\left[P \leq \frac{n - \sqrt{n \ln n}}{2\phi}\right] \leq \mathbf{Pr}\left[X \leq \frac{n - \sqrt{n \ln n}}{2\phi}\right] \\ &= \mathbf{Pr}\left[\mathbf{E}[X] - X \geq \frac{\sqrt{n \ln n}}{2\phi}\right] \leq e^{-(\ln n)/2} = \frac{1}{\sqrt{n}}\,, \end{aligned} \tag{3.3}$$

where the last inequality follows from Hoeffding's bound [67]. Consider the random variable

$$Z = \begin{cases} \frac{1}{2} + \sqrt{n} & \text{if event } \mathcal{F} \text{ occurs}\,, \\ 1 + \frac{n-1}{P} & \text{otherwise}\,, \end{cases}$$

and let $Y = \max_{\sigma(\mathcal{I}) \in \mathrm{Jump}(\mathcal{I})}(C_{\max}^{\sigma(\mathcal{I})}/C_{\max}^{OPT(\mathcal{I})})$. Inequalities (3.1) and (3.2) yield $Y \leq Z$. Let $\overline{\mathcal{F}}$ be the complement of $\mathcal{F}$. Then,

$$\begin{aligned}\mathop{\mathbf{E}}_{\mathcal{I}\sim\Phi}[Y] &\leq \mathop{\mathbf{E}}_{\mathcal{I}\sim\Phi}[Z] \leq \mathop{\mathbf{E}}_{\mathcal{I}\sim\Phi}\left[Z \middle| \overline{\mathcal{F}}\right] + \mathop{\mathbf{E}}_{\mathcal{I}\sim\Phi}[Z|\,\mathcal{F}] \cdot \mathop{\mathbf{Pr}}_{\mathcal{I}\sim\Phi}[\mathcal{F}] \\ &\overset{(3.3)}{\leq} \left(1 + \frac{2\phi(n-1)}{n - \sqrt{n \ln n}}\right) + \frac{1/2 + \sqrt{n}}{\sqrt{n}} < 2.5 + \frac{2\phi}{1 - \sqrt{\ln(n)/n}} < 2.5 + 5.1\phi\,. \end{aligned}$$

The third inequality follows since $P > (n - \sqrt{n \ln n})/(2\phi)$ if event $\mathcal{F}$ does not hold. The last inequality holds as

$$\max_{n \in \mathbb{Z}^+} \frac{2\phi}{1 - \sqrt{\ln(n)/n}} < 5.1,$$

where the maximum is attained for $n = 3$. □

Corollary 3.2.3. *Consider an instance for related parallel machines in which the processing requirement of each job is chosen independently and uniformly at random from* $[0,1]$. *The expected performance guarantee of the worst jump optimal schedule is bounded by a constant.*

I proceed with showing that the upper bound on the smoothed performance guarantee provided in Theorem 3.2.2 is tight up to constant factor when $\phi \geq 2$.

Theorem 3.2.4. *There is a class of ϕ-smooth instances Φ with related parallel machines such that*

$$\mathop{\mathbf{E}}_{\mathcal{I}\sim\Phi}\left[\max_{\sigma(\mathcal{I}) \in \mathrm{Jump}(\mathcal{I})} \frac{C_{\max}^{\sigma(\mathcal{I})}}{C_{\max}^{OPT(\mathcal{I})}}\right] = \Omega(\phi)\,.$$

Proof. For any $\phi \geq 2$, construct a ϕ-smooth instance Φ with $n = \lceil 4\phi^2 + 1 \rceil$ and $m = n$ machines. Let

$$s_1 = \frac{n-1}{4\phi} \geq \phi \geq 2 \quad \text{and} \quad s_2 = \ldots = s_n = 1\,.$$

Let the processing requirement p_1 be chosen uniformly from the interval $[1 - 1/\phi, 1]$ while the processing requirements of all other jobs are chosen uniformly from the interval $[0, 1/\phi]$. Let

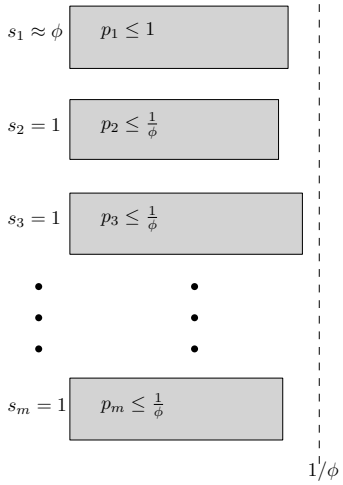

Figure 3.1: The optimal schedule $OPT(\mathcal{I})$.

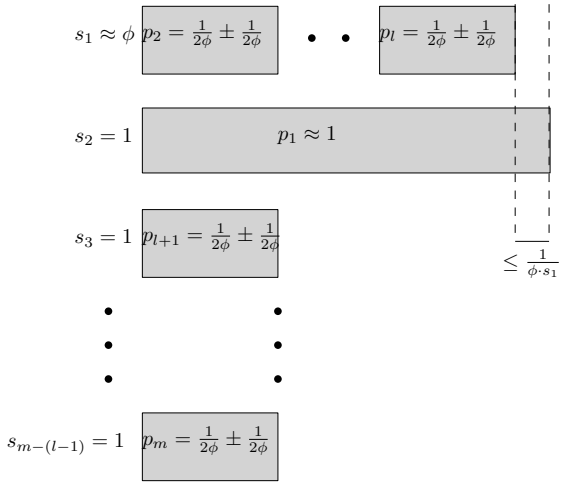

Figure 3.2: The non-empty machines in schedule $\sigma(\mathcal{I})$ if $\mathcal{E}$ occurs.

$\mathcal{I}$ be any realization of Φ. In an optimal schedule $OPT(\mathcal{I})$, job 1 is scheduled on machine 1, and all other machines process exactly one job, see Figure 3.1. Hence,

$$C_{\max}^{OPT(\mathcal{I})} = \max\left\{\frac{p_1}{s_1}, p_2, \ldots, p_n\right\} \leq \max\left\{\frac{1}{s_1}, \frac{1}{\phi}\right\} = \frac{1}{\phi}\,.$$

I show that with high probability there exists a jump optimal schedule $\sigma(\mathcal{I})$ with $C_{\max}^{\sigma(\mathcal{I})} > 1 - 1/\phi$. In order to find such a schedule $\sigma(\mathcal{I})$, first schedule job 1 on machine 2. Then, consider the remaining jobs one after another (like list scheduling) and schedule unassigned jobs to machine 1 until either $L_1^{\sigma(\mathcal{I})} \in \left[L_2^{\sigma(\mathcal{I})} - 1/(\phi s_1), L_2^{\sigma(\mathcal{I})}\right)$ or all jobs are scheduled. Any job that remains unscheduled is then exclusively assigned to one empty machine. Let $\mathcal{E}$ denote the event that $Q := \sum_{j=2}^{n} p_j \geq s_1$. Note that $\mathbf{E}[Q] = (n-1)/(2\phi) = 2s_1$. It will be shown that event $\mathcal{E}$ holds with high probability with respect to ϕ.

Consider the case that event $\mathcal{E}$ occurs. Then, schedule $\sigma(\mathcal{I})$ is such that $L_1^{\sigma(\mathcal{I})} \in \left[L_2^{\sigma(\mathcal{I})} - 1/(\phi s_1), L_2^{\sigma(\mathcal{I})}\right)$ since $Q/s_1 \geq 1 \geq p_1 = L_2^{\sigma(\mathcal{I})}$ and $p_j \leq 1/\phi$ for all jobs $j = 2, \ldots, n$, see Figure 3.2. Next, I argue that schedule $\sigma(\mathcal{I})$ is jump optimal. First observe that machine 2 defines the makespan. Job 1, which is the only job assigned to that machine, cannot jump to a machine $i > 2$ as these have the same speed as machine 2. Furthermore, job 1 cannot jump to machine 1 because

$$L_1^{\sigma(\mathcal{I})} + \frac{p_1}{s_1} \geq L_2^{\sigma(\mathcal{I})} - \frac{1}{\phi s_1} + \frac{1 - 1/\phi}{s_1} = L_2^{\sigma(\mathcal{I})} + \frac{1 - 2/\phi}{s_1} \geq L_2^{\sigma(\mathcal{I})}$$

as $\phi \geq 2$. Hence, $\sigma(\mathcal{I})$ is a jump optimal schedule with

$$\frac{C_{\max}^{\sigma(\mathcal{I})}}{C_{\max}^{OPT(\mathcal{I})}} > \frac{1 - 1/\phi}{1/\phi} = \phi - 1\,. \tag{3.4}$$

It remains to determine the probability of event $\mathcal{E}$. Recalling $\mathbf{E}[Q] = 2s_1$, $s_1 = (n-1)/(4\phi)$, and $n \geq 4\phi^2 + 1$, the probability of the counterevent $\overline{\mathcal{E}}$ can be bounded with Hoeffding's bound [67] as follows:

$$\mathbf{Pr}\left[\overline{\mathcal{E}}\right] = \mathbf{Pr}\left[Q < s_1\right] = \mathbf{Pr}\left[\mathbf{E}[Q] - Q > s_1\right] \leq e^{\frac{-2s_1^2}{(n-1)/\phi^2}} = e^{\frac{-2\left(\frac{n-1}{4\phi}\right)^2}{(n-1)/\phi^2}} = e^{-(n-1)/8} \leq e^{-\phi^2/2}.$$

Let $X = \max_{\sigma(\mathcal{I}) \in \mathrm{Jump}(\mathcal{I})}(C_{\max}^{\sigma(\mathcal{I})}/C_{\max}^{OPT(\mathcal{I})})$. Applying Inequality (3.4) the smoothed performance guarantee can be bounded from below as follows:

$$\begin{aligned}\mathop{\mathbf{E}}_{\mathcal{I}\sim\Phi}[X] &\geq \mathop{\mathbf{E}}_{\mathcal{I}\sim\Phi}[X \,|\, \mathcal{E}] \cdot \mathop{\mathbf{Pr}}_{\mathcal{I}\sim\Phi}[\mathcal{E}] \geq (\phi-1)\cdot\left(1 - e^{-\phi^2/2}\right) \\ &= (\phi-1) - (\phi-1)\cdot e^{-\phi^2/2} > \phi - 1.14 = \Omega(\phi),\end{aligned}$$

where the last inequality follows because $(\phi-1)\cdot e^{-\phi^2/2} < 0.14$ for $\phi \geq 2$. □

3.2.2 Upper bounds for lexjump optimal schedules

This subsection shows that the smoothed performance guarantee of lexjump scheduling is in $O(\log\phi)$. The next subsection shows this bound to be asymptotically tight. Let $\mathrm{Lex}(\mathcal{I})$ represent the set of all possible lexjump optimal schedules for some instance $\mathcal{I}$. Recall that by definition, it holds for any lexjump optimal schedule $\sigma(\mathcal{I}) \in \mathrm{Lex}(\mathcal{I})$, for any two machines i and i', and for any job $j \in J_i^{\sigma(\mathcal{I})}$ that

$$L_{i'}^{\sigma(\mathcal{I})} + \frac{p_j}{s_{i'}} \geq L_i^{\sigma(\mathcal{I})}. \tag{3.5}$$

The main theorem is given below.

Theorem 3.2.5. *Let β be an arbitrary positive real. For $\phi \geq 2$ and any ϕ-smooth instance Φ with related parallel machines*

$$\mathop{\mathbf{Pr}}_{\mathcal{I}\sim\Phi}\left[\max_{\sigma(\mathcal{I})\in\mathrm{Lex}(\mathcal{I})} \frac{C_{\max}^{\sigma(\mathcal{I})}}{C_{\max}^{OPT(\mathcal{I})}} \geq \beta\right] \leq \left(\frac{32\phi}{2^{\beta/3}}\right)^{n/2}$$

and

$$\mathop{\mathbf{E}}_{\mathcal{I}\sim\Phi}\left[\max_{\sigma(\mathcal{I})\in\mathrm{Lex}(\mathcal{I})} \frac{C_{\max}^{\sigma(\mathcal{I})}}{C_{\max}^{OPT(\mathcal{I})}}\right] \leq 9\log\phi + 15 = O(\log\phi).$$

Note that the assumption $\phi \geq 2$ in Theorem 3.2.5 is no real restriction as for $\phi \in [1,2)$, any ϕ-smooth instance is a 2-smooth instance as well. Hence, for these values all bounds from Theorem 3.2.5 can be applied, when substituting ϕ by 2. In particular, the expected value is a constant.

The first part of this subsection derives a series of structural properties which hold for any lexjump optimal schedule $\sigma(\mathcal{I})$ on any (non-smoothed) instance $\mathcal{I}$. The main insight to be gained is that in case the ratio of $C_{\max}^{\sigma(\mathcal{I})}$ over $C_{\max}^{OPT(\mathcal{I})}$ is large for some instance $\mathcal{I}$ and schedule $\sigma(\mathcal{I}) \in \mathrm{Lex}(\mathcal{I})$, then the instance needs to have many very small jobs. This holds even when the instance $\mathcal{I}$ is deterministically picked by some adversary. I show this property

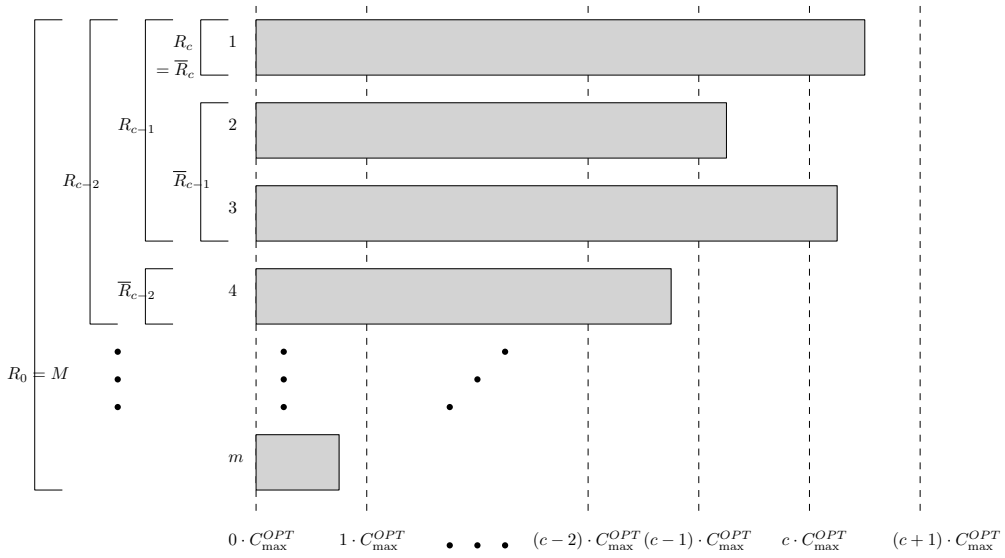

Figure 3.3: Definition of R_k and $\overline{R}_k$.

in Property 3.2.6 up to Lemma 3.2.18. This observation allows us to prove the main theorem of this subsection by showing that for any ϕ-smooth instance, there are only a few such small jobs with high probability. The latter implies that a large ratio only happens with very small probability.

In the proofs below, I adopt notation which was introduced by Czumaj and Vöcking [38]. Order the machines such that $s_1 \geq s_2 \geq \ldots \geq s_m$. Given an instance $\mathcal{I}$ and a corresponding lexjump optimal schedule $\sigma(\mathcal{I})$ and optimal schedule $OPT(\mathcal{I})$, define $c = \left\lfloor C_{\max}^{\sigma(\mathcal{I})} / C_{\max}^{OPT(\mathcal{I})} \right\rfloor - 1$. Assume c to be big enough, i. e., $c \geq 3$. Define the set $R_k^{\sigma(\mathcal{I})}$, or just R_k for short, to be the set $R_k = \{1, \ldots, i_k\}$ where $i_k = \max\left\{i \in M : L_h^{\sigma(\mathcal{I})} \geq k \cdot C_{\max}^{OPT(\mathcal{I})} \, \forall h \leq i\right\}$. Note that $i_k = m$ for all $k \leq 0$ and hence $R_k = M$ for such k (see Figure 3.3). Further, define $\overline{R}_k^{\sigma(\mathcal{I})}$, or just $\overline{R}_k$ for short, to be the set $R_k \setminus R_{k+1}$, for all $k \in \{0, \ldots, c-1\}$. Also, let $\overline{R}_c = R_c$. Note that this classification always refers to schedule $\sigma(\mathcal{I})$ even if additionally other schedules are considered. Further, observe that although the notation is equal, the definition of the sets R_k and $\overline{R}_k$ is similar but not identical to their definitions in the previous chapter in Subsection 2.3. Some properties follow straightforwardly.

Property 3.2.6. *For each machine $i \in R_k$, $L_i^{\sigma(\mathcal{I})} \geq k \cdot C_{\max}^{OPT(\mathcal{I})}$.*

Property 3.2.7. *Machine $i_k + 1$ is the fastest machine in $M \setminus R_k$, for all $k \in \{1, \ldots, c\}$.*

Property 3.2.8. $L_{i_k+1}^{\sigma(\mathcal{I})} < k \cdot C_{\max}^{OPT(\mathcal{I})}$ *for all* $k \in \{1, \ldots, c\}$.

Property 3.2.9. $L_1^{\sigma(\mathcal{I})} < (c+2) \cdot C_{\max}^{OPT(\mathcal{I})}$.

Lemma 3.2.10. *Machine* 1 *is in class* $R_c = \overline{R}_c$.

Proof. If machine 1 is a critical machine, then $L_1^{\sigma(\mathcal{I})}/C_{\max}^{\sigma(\mathcal{I})} = C_{\max}^{\sigma(\mathcal{I})}/C_{\max}^{\sigma(\mathcal{I})} > c$ which yields that machine 1 is in $R_c = \overline{R}_c$. Otherwise, there exists another machine i which is critical. Consider any job $j \in J_i^{\sigma(\mathcal{I})}$. Since machine i is critical it follows by definition of c that $L_i^{\sigma(\mathcal{I})} = C_{\max}^{\sigma(\mathcal{I})} \geq (c+1)C_{\max}^{OPT(\mathcal{I})}$. Applying inequality (3.5) for job $j \in J_i^{\sigma(\mathcal{I})}$ and machine 1 yields $L_1^{\sigma(\mathcal{I})} + p_j/s_1 \geq L_i^{\sigma(\mathcal{I})}$. Hence,

$$L_1^{\sigma(\mathcal{I})}/C_{\max}^{OPT(\mathcal{I})} \geq L_i^{\sigma(\mathcal{I})}/C_{\max}^{OPT(\mathcal{I})} - (p_j/s_1)/C_{\max}^{OPT(\mathcal{I})} \geq L_i^{\sigma(\mathcal{I})}/C_{\max}^{OPT(\mathcal{I})} - 1 \geq c,$$

where the second inequality is due to the fact that any job j can contribute at most $C_{\max}^{OPT(\mathcal{I})}$ to the load of the fastest machine (which is machine 1). I conclude that $L_1^{\sigma(\mathcal{I})}/C_{\max}^{OPT(\mathcal{I})} \geq c$ such that machine 1 in $R_c = \overline{R}_c$. □

The previous Lemma established that the class $\overline{R}_c$ is nonempty. Next, I show that none of the classes $\overline{R}_k$, for all $k \in \{0, \ldots, c\}$, will be empty.

Lemma 3.2.11. *There is at least one job which is being processed on a machine in* $\overline{R}_k$ *in lexjump optimal schedule* $\sigma(\mathcal{I})$*, which is in an optimal schedule* $OPT(\mathcal{I})$ *assigned on a machine in* $\overline{R}_{k-1}$*, for all* $k \in \{1, 2, \ldots, c\}$.

Proof. First I show the statement for $k = c$. Lemma 3.2.10 established that the set $\overline{R}_c$ is nonempty. For all machines $i \in \overline{R}_c$, Property 3.2.6 gives $L_i^{\sigma(\mathcal{I})} \geq c \cdot C_{\max}^{OPT(\mathcal{I})} > C_{\max}^{OPT(\mathcal{I})}$, as $c \geq 3$. Hence, there exists a job $j \in J_i^{\sigma(\mathcal{I})}$ for some machine $i \in \overline{R}_c$ which is scheduled on a machine in $M \backslash R_c$ in an optimal schedule $OPT(\mathcal{I})$. Assume that the set $M \backslash R_{c-1}$ is non-empty since otherwise job j needs to be assigned to a machine in $\overline{R}_{c-1}$ proving the statement. Let $i^* = i_{c-1} + 1$ be the fastest machine in $M \backslash R_{c-1}$. Property 3.2.8 yields that $L_{i^*}^{\sigma(\mathcal{I})} < (c-1) \cdot C_{\max}^{OPT(\mathcal{I})}$. Combining this insight with the lexjump optimality of schedule $\sigma(\mathcal{I})$ and Property 3.2.6 yields

$$(c-1) \cdot C_{\max}^{OPT(\mathcal{I})} + \frac{p_j}{s_{i^*}} > L_{i^*}^{\sigma(\mathcal{I})} + \frac{p_j}{s_{i^*}} \geq L_i^{\sigma(\mathcal{I})} \geq c \cdot C_{\max}^{OPT(\mathcal{I})}.$$

That is, $p_j/s_h \geq p_j/s_{i^*} > C_{\max}^{OPT(\mathcal{I})}$ for all machines $h \in M \backslash R_{c-1}$ and hence job j needs to be assigned to a machine in $\overline{R}_{c-1}$ in an optimal schedule $OPT(\mathcal{I})$. Therefore, the set $\overline{R}_{c-1}$ cannot be empty.

The preceding line of reasoning can be iterated for all $k \in \{2, \ldots, c\}$. Finally, the statement holds also true for $k = 1$ since there exists a machine $i \in \overline{R}_1$ which in an optimal schedule $OPT(\mathcal{I})$ gets assigned a job from $\bigcup_{h \in \overline{R}_2} J_h^{\sigma(\mathcal{I})}$. As $L_i^{\sigma(\mathcal{I})} \geq C_{\max}^{OPT(\mathcal{I})}$ it follows that there needs to be a job $j \in J_i^{\sigma(\mathcal{I})}$ which is assigned to a machine in $\overline{R}_0$ in an optimal schedule $OPT(\mathcal{I})$. Hence, $\overline{R}_0$ needs to be non-empty. □

Corollary 3.2.12. $\overline{R_k} \neq \emptyset$ *for all* $k \in \{0, 1, 2, \ldots, c\}$.

The proof of Lemma 3.2.11 implies that a job scheduled to a machine in R_k in a lexjump optimal schedule $\sigma(\mathcal{I})$ cannot be scheduled to a machine in $M \backslash R_{k-1}$ in an optimal schedule $OPT(\mathcal{I})$.

Corollary 3.2.13. *Let $\sigma(\mathcal{I})$ be a lexjump optimal schedule and let R_k be defined accordingly. Then, a job $j \in J_i^{\sigma(\mathcal{I})}$ for a machine $i \in R_k$ will in an optimal schedule $OPT(\mathcal{I})$ be scheduled to a machine in R_{k-1}.*

The above statement can be generalized to show how much a job j currently scheduled to a machine in R_{k_1} contributes to the load of a machine in $M \setminus R_{k_2}$, where $k_2 < k_1$. In particular, Lemma 3.2.14 implies Corollary 3.2.13 if one takes $k_2 = k_1 - 1$.

Lemma 3.2.14. *Let $0 < k_2 < k_1 \leq c$ be positive integers, let i_1 and i_2 be machines such that $i_1 \in R_{k_1}$ and $i_2 \in M \setminus R_{k_2}$. Also, let job j be a job on machine i_1. Then, the processing time of job j for machine i_2 is lower bounded by $p_j/s_{i_2} > (k_1 - k_2) \cdot C_{\max}^{OPT(\mathcal{I})}$.*

Proof. Let $i^* = i_{k_2} + 1$ be the fastest machine in $M \setminus R_{k_2}$. By jump optimality of $\sigma(\mathcal{I})$ it follows that $L_{i_1}^{\sigma(\mathcal{I})} \leq L_{i^*}^{\sigma(\mathcal{I})} + p_j/s_{i^*}$. Then, Property 3.2.6 for machine i_1 and Property 3.2.8 for machine i^* yield

$$\frac{p_j}{s_{i^*}} \geq L_{i_1}^{\sigma(\mathcal{I})} - L_{i^*}^{\sigma(\mathcal{I})} > L_{i_1}^{\sigma(\mathcal{I})} - k_2 C_{\max}^{OPT(\mathcal{I})} \geq (k_1 - k_2) C_{\max}^{OPT(\mathcal{I})}.$$

It follows that for all machines $i_2 \in M \setminus R_{k_2}$ it is true that $p_j/s_{i_2} \geq p_j/s_{i^*} > (k_1 - k_2) C_{\max}^{OPT(\mathcal{I})}$. □

Next, I show that in a lexjump optimal schedule the speeds of any two machines which are at least three classes apart differ by a factor of at least 2. Czumaj and Vöcking [38] and Aspnes et al. [5] showed similar properties.

Lemma 3.2.15. *The speed of any machine in class R_k, for $k \in \{3, \ldots, c\}$, is at least twice the speed of any machine in $M \setminus R_{k-2}$.*

Proof. For $k \geq 3$ the set $M \setminus R_{k-2}$ contains the set $\overline{R}_0$. Therefore, by Corolloray 3.2.12, $M \setminus R_{k-2} \neq \emptyset$. Lemma 3.2.11 shows that there exists a job j scheduled on a machine in the class R_k in schedule $\sigma(\mathcal{I})$, which will be scheduled on a machine $i_2 \in \overline{R}_{k-1}$ in an optimal schedule $OPT(\mathcal{I})$. Therefore, $p_j \leq s_{i_2} C_{\max}^{OPT(\mathcal{I})} \leq s_{i_1} C_{\max}^{OPT(\mathcal{I})}$, where i_1 represents any machine in R_k. Let machine i_3 be any machine in the set $M \setminus R_{k-2}$. Then, Lemma 3.2.14 yields $p_j/s_{i_3} > (k - (k-2)) \cdot C_{\max}^{OPT(\mathcal{I})} = 2C_{\max}^{OPT(\mathcal{I})}$. It follows that $s_{i_1} C_{\max}^{OPT(\mathcal{I})} > 2 s_{i_3} C_{\max}^{OPT(\mathcal{I})}$, i. e., $s_{i_1} > 2 s_{i_3}$. □

Lemma 3.2.16. *Let $0 \leq k_2 \leq k_1 \leq c$ be two integers, let i_1 and i_2 be two machines such that $i_1 \in R_{k_1}$ and $i_2 \in R_{k_2}$. Then, $s_{i_1} \geq s_{i_2} \cdot 2^{\lfloor \Delta/3 \rfloor}$ where $\Delta = k_1 - k_2$.*

Proof. The claim is proven by induction. For $\Delta \in \{0, \ldots, 2\}$, the claim trivially holds as $s_{i_1} \geq s_{i_2}$ and as the machines are ordered to their speeds. Assume that the claim holds up to some integer Δ^* where $\Delta^* \geq 2$. Then, I show that it is also true for $\Delta = \Delta^* + 1 \geq 3$. According to Corollary 3.2.12 the class $R_{k_1-3} \setminus R_{k_1-2} = \overline{R}_{k_1-3}$ contains at least one machine. Let $i^* = i_{k_1-2} + 1$ be the fastest machine in $\overline{R}_{k_1-3}$. Then,

$$s_{i_1} \geq 2 s_{i^*} \geq 2 \cdot s_{i_2} \cdot 2^{\lfloor (\Delta - 3)/3 \rfloor} = s_{i_2} \cdot 2^{\lfloor \Delta/3 \rfloor},$$

where the first inequality follows by Lemma 3.2.15, and the second inequality by induction. □

Since the machines in low classes are exponentially slower than the machines in the highest two classes, the jobs assigned to these machines in an optimal schedule $OPT(\mathcal{I})$ have processing requirements exponentially small in c.

Lemma 3.2.17. *Let i be a machine in $M \setminus R_2 = \overline{R}_1 \cup \overline{R}_0$. Then each job j assigned to machine i in an optimal schedule has processing requirement at most $p_j \leq 2^{-c/3+2}$.*

Proof. For $c \leq 6$ the claim is true since all processing requirements have been rescaled to be at most one. Assume $c \geq 7$. Consider an optimal schedule $OPT(\mathcal{I})$ and let j be a job processed on a machine $i \in M \setminus R_2$ according to $OPT(\mathcal{I})$. By Corollary 3.2.12, $M \setminus R_2 = \overline{R}_1 \cup \overline{R}_0 \neq \emptyset$. Then, $p_j/s_i \leq C_{\max}^{OPT(\mathcal{I})}$, i. e.,

$$p_j \leq s_i \cdot C_{\max}^{OPT(\mathcal{I})}. \tag{3.6}$$

To bound $s_i \cdot C_{\max}^{OPT(\mathcal{I})}$, consider a job j' on machine 1 of lexjump optimal schedule $\sigma(\mathcal{I})$ and consider the first machine $i' \in \overline{R}_{c-3} \neq \emptyset$. Applying Inequality (3.5), it follows that $L_{i'}^{\sigma(\mathcal{I})} + p_{j'}/s_{i'} \geq L_1^{\sigma(\mathcal{I})}$, i.e., $p_{j'} \geq s_{i'} \cdot (L_1^{\sigma(\mathcal{I})} - L_{i'}^{\sigma(\mathcal{I})})$. Since machine 1 belongs to R_c due to Property 3.2.6 and since machine i' is the first machine that does not belong to R_{c-2}, it holds that $L_1^{\sigma(\mathcal{I})} \geq c \cdot C_{\max}^{OPT(\mathcal{I})}$ and $L_{i'}^{\sigma(\mathcal{I})} < (c-2) \cdot C_{\max}^{OPT(\mathcal{I})}$, which implies $p_{j'} \geq 2 s_{i'} \cdot C_{\max}^{OPT(\mathcal{I})}$. Lemma 3.2.16 yields $s_{i'} \geq s_i \cdot 2^{\lfloor (c-4)/3 \rfloor}$. Applying Inequality 3.6 and $p_{j'} \leq 1$, I obtain

$$p_j \leq s_i \cdot C_{\max}^{OPT(\mathcal{I})} \leq s_{i'} \cdot C_{\max}^{OPT(\mathcal{I})} \cdot 2^{-\lfloor (c-4)/3 \rfloor} < p_{j'} \cdot 2^{-c/3+4/3} < 2^{-c/3+2}.$$

□

Lemma 3.2.18. *The processing requirement of at least $n/2$ jobs is at most $2^{-c/3+2}$.*

Proof. In the lexjump optimal schedule $\sigma(\mathcal{I})$ the load of any machine $i \in R_2$ is at least $2C_{\max}^{OPT(\mathcal{I})}$ whereas in the optimal schedule $OPT(\mathcal{I})$ the load of such a machine can be at most $C_{\max}^{OPT(\mathcal{I})}$. Therefore, in the lexjump optimal schedule $\sigma(\mathcal{I})$, each machine $i \in R_2$ processes jobs who will be assigned to machines $M \setminus R_2$ in an optimal schedule $OPT(\mathcal{I})$. The load of these jobs per machine will be at least $C_{\max}^{OPT(\mathcal{I})}$. Hence, a total processing requirement of at least $\sum_{i \in R_2} s_i \cdot C_{\max}^{OPT(\mathcal{I})}$, which is assigned in schedule $\sigma(\mathcal{I})$ to machines in R_2, is scheduled on machines in $M \setminus R_2 = \overline{R}_1 \cup \overline{R}_0$ in an optimal schedule $OPT(\mathcal{I})$. Since the amount $\sum_{i \in R_2} s_i \cdot C_{\max}^{OPT(\mathcal{I})}$ simultaneously is an upper bound on the total processing requirement which the machines in R_2 can process in an optimal schedule $OPT(\mathcal{I})$, it follows that in $OPT(\mathcal{I})$, at least half of the total processing requirement is processed by machines in $\overline{R}_1 \cup \overline{R}_0$. Since all jobs scheduled on machines in $M \setminus R_2$ by an optimal schedule $OPT(\mathcal{I})$ have a processing requirement of at most $2^{-c/3+2}$, see Lemma 3.2.17, at least half of the jobs have a processing requirement of at most $2^{-c/3+2}$. □

Since having many such small jobs is unlikely when the processing requirements have been smoothed, it follows that the smoothed performance guarantee, which is in between c and $c+1$, cannot be too high, yielding Theorem 3.2.5.

Proof of Theorem 3.2.5. If $C_{\max}^{\sigma(\mathcal{I})}/C_{\max}^{OPT(\mathcal{I})} \geq \beta$, then at least $n/2$ jobs have processing requirement at most $2^{-\beta/3+3}$ due to Lemma 3.2.17 and $c = \left\lfloor C_{\max}^{\sigma(\mathcal{I})}/C_{\max}^{OPT(\mathcal{I})} \right\rfloor - 1 \geq \beta - 2$. The probability that one specific job is such small is bounded by $\phi \cdot 2^{-\beta/3+3} = 8\phi \cdot 2^{-\beta/3}$ in the smoothed

input model. Hence, the probability that the processing requirement of at least $n/2$ jobs is at most $2^{-\beta/3+3}$, is bounded by

$$\sum_{k\geq\frac{n}{2}}\binom{n}{k}\left(8\phi\cdot 2^{-\beta/3}\right)^k\cdot\left(1-8\phi\cdot 2^{-\beta/3}\right)^{n-k}\leq\sum_{k\geq\frac{n}{2}}\binom{n}{k}\left(8\phi\cdot 2^{-\beta/3}\right)^{n/2}$$
$$\leq 2^n\cdot\left(8\phi\cdot 2^{-\beta/3}\right)^{n/2}=\left(32\phi\cdot 2^{-\beta/3}\right)^{n/2}\,.$$

Note that the first inequality holds if $8\phi\cdot 2^{-\beta/3}<1$. Otherwise, the bound holds true trivially. This yields

$$\mathop{\mathbf{Pr}}_{\mathcal{I}\sim\Phi}\left[\max_{\sigma(\mathcal{I})\in\mathrm{Lex}(\mathcal{I})}\frac{C_{\max}^{\sigma(\mathcal{I})}}{C_{\max}^{OPT(\mathcal{I})}}\geq\beta\right]\leq\left(\frac{32\phi}{2^{\beta/3}}\right)^{n/2}\,.$$

As for $n=1$ any schedule $\sigma(\mathcal{I})\in\mathrm{Lex}(\mathcal{I})$ is optimal, it is safe to just consider the case $n\geq 2$. For $k\geq 1$ let $\beta_k=\beta_k(\phi)=3k\log_2\phi+15$, i.e., $2^{\beta_k/3}=32\phi^k$. If $\beta\geq\beta_k$, then

$$\mathop{\mathbf{Pr}}_{\mathcal{I}\sim\Phi}\left[\max_{\sigma(\mathcal{I})\in\mathrm{Lex}(\mathcal{I})}\frac{C_{\max}^{\sigma(\mathcal{I})}}{C_{\max}^{OPT(\mathcal{I})}}\geq\beta\right]\leq\mathop{\mathbf{Pr}}_{\mathcal{I}\sim\Phi}\left[\max_{\sigma(\mathcal{I})\in\mathrm{Lex}(\mathcal{I})}\frac{C_{\max}^{\sigma(\mathcal{I})}}{C_{\max}^{OPT(\mathcal{I})}}\geq\beta_k\right]$$
$$\leq\left(\phi^{1-k}\right)^{n/2}\leq\phi^{1-k}\leq 2^{1-k}$$

as $\phi\geq 2$. Since $\beta_{k+1}-\beta_k=3\log_2\phi$ it is that

$$\mathop{\mathbf{E}}_{\mathcal{I}\sim\Phi}\left[\max_{\sigma(\mathcal{I})\in\mathrm{Lex}(\mathcal{I})}\frac{C_{\max}^{\sigma(\mathcal{I})}}{C_{\max}^{OPT(\mathcal{I})}}\right]=\int_0^\infty\mathop{\mathbf{Pr}}_{\mathcal{I}\sim\Phi}\left[\max_{\sigma(\mathcal{I})\in\mathrm{Lex}(\mathcal{I})}\frac{C_{\max}^{\sigma(\mathcal{I})}}{C_{\max}^{OPT(\mathcal{I})}}\geq\beta\right]\mathrm{d}\beta$$
$$\leq\beta_1+\sum_{k=1}^{\infty}\int_{\beta_k}^{\beta_{k+1}}\mathop{\mathbf{Pr}}_{\mathcal{I}\sim\Phi}\left[\max_{\sigma(\mathcal{I})\in\mathrm{Lex}(\mathcal{I})}\frac{C_{\max}^{\sigma(\mathcal{I})}}{C_{\max}^{OPT(\mathcal{I})}}\geq\beta\right]\mathrm{d}\beta$$
$$\leq\beta_1+3\log_2\phi\cdot\sum_{k=1}^{\infty}2^{1-k}=9\log_2\phi+15\,.$$

□

3.2.3 Lower bounds for lexjump optimal schedules

The following theorem shows that the upper bound provided in Theorem 3.2.5 is tight up to a constant factor.

Theorem 3.2.19. *There is a class of ϕ-smooth instances Φ with related parallel machines such that, for any $\mathcal{I}\in\Phi$,*

$$\max_{\sigma(\mathcal{I})\in\mathrm{Lex}(\mathcal{I})}\frac{C_{\max}^{\sigma(\mathcal{I})}}{C_{\max}^{OPT(\mathcal{I})})}=\Omega(\log\phi)\quad.$$

Note that the result stated above holds with probability 1.

This subsection is structured as follows: First, I present a ϕ-smooth instance Φ along with a permutation of the jobs. Afterwards, I analyze the schedule $\sigma(\mathcal{I})$ which will be returned by the list scheduling algorithm given the provided permutation of the jobs, for any instance

$\mathcal{I} \sim \Phi$. The schedule $\sigma(\mathcal{I})$ resembles the worst-case example constructed by Czumaj and Vöcking [38]. Finally, I show that $\sigma(\mathcal{I})$ is lexjump optimal yielding a lower bound of $\Omega(\log \phi)$ for the worst lexjump optimal schedules on related parallel machines.

As scaling of all processing requirements does not change the approximation ratio, for sake of simplicity, I do not consider probability densities $f_j \colon [0,1] \to [0,\phi]$ but scaled densities $f'_j \colon [0, 2^{r+1}] \to [0, \phi/2^{r+1}]$ for an appropriate integer r.

Let $\phi \geq 4$ and consider an integer $r = \lfloor \log_4 \phi \rfloor \geq 1$, i.e., $\phi \geq 4^r = 2^{2r}$. The machines are partitioned into machine classes M_k for $k = 0, \dots, r$, such that machine class M_k contains $r!/k!$ machines of speed 2^k. Let the instance $\mathcal{I}$ be any realization of the ϕ-smooth instance Φ. Jobs are partitioned in job classes J_l for $l = 1, \dots, r$ such that a job class J_l contains $r!/(l-1)!$ jobs each having a processing requirement uniformly drawn from $\left[2^l, 2^l + 2^{r+1}/\phi\right) \subseteq (0, 2^{r+1})$. Note that the density of this instance is bounded by $\phi/2^{r+1}$ which is valid in our model. Algorithm LB-LIST, which is depicted below, shows the permutation of the jobs such that schedule $\sigma(\mathcal{I})$ is constructed when scheduling the jobs using list scheduling.

Algorithm 1 Algorithm Lowerbound-List-Scheduling (LB-LIST)

Require: An instance $\mathcal{I}$ with job classes J_l and machine classes M_l, $l \in [r]$.

1: **for** $k = 1$ **to** r **do**
2: **for** $l = r$ **down to** k **do**
3: Schedule $r!/l!$ arbitrary jobs of class J_l to machines in M_l, according to list scheduling.
4: **end for**
5: **end for**
6: **Return** schedule $\sigma(\mathcal{I})$.

Note that for any job class J_l all $l \cdot r!/l! = r!/(l-1)!$ jobs have been scheduled. Let $\sigma(\mathcal{I})$ be the schedule returned by LB-LIST. First a key property of $\sigma(\mathcal{I})$ will be shown.

Lemma 3.2.20. *For any index $l = 1, \dots, r$ each machine in M_l is assigned exactly l jobs of job class J_l and no other jobs. The machines in M_0 remain empty.*

Proof. Let $\sigma(k,l) = \sigma(\mathcal{I}, k, l)$ denote the partial schedule after processing step 3 of iteration (k,l) of Algorithm LB-LIST. Within the $(k,l)^{th}$ iteration, a machine $i \in M_l$ is called *used* if a job of class J_l has already been assigned to i during that iteration. Otherwise, the machine i is *unused*. The following two claims are shown inductively and simultaneously. The Lemma then follows straightforwardly from the second claim since the last iteration is (r,r).

Claim 3.2.21. *During iteration (k,l), $r!/l!$ jobs of class J_l are assigned to $r!/l!$ distinct machines, i. e., all machines, of class M_l.*

Claim 3.2.22. *In the partial schedule $\sigma(k,l)$ each machine in class $M_{l'}$ is assigned*

$$k' = \begin{cases} k & : \quad l' \geq l\,, \\ \min\{k-1, l'\} & : \quad l' < l\,, \end{cases}$$

jobs of class $J_{l'}$ and no other jobs.

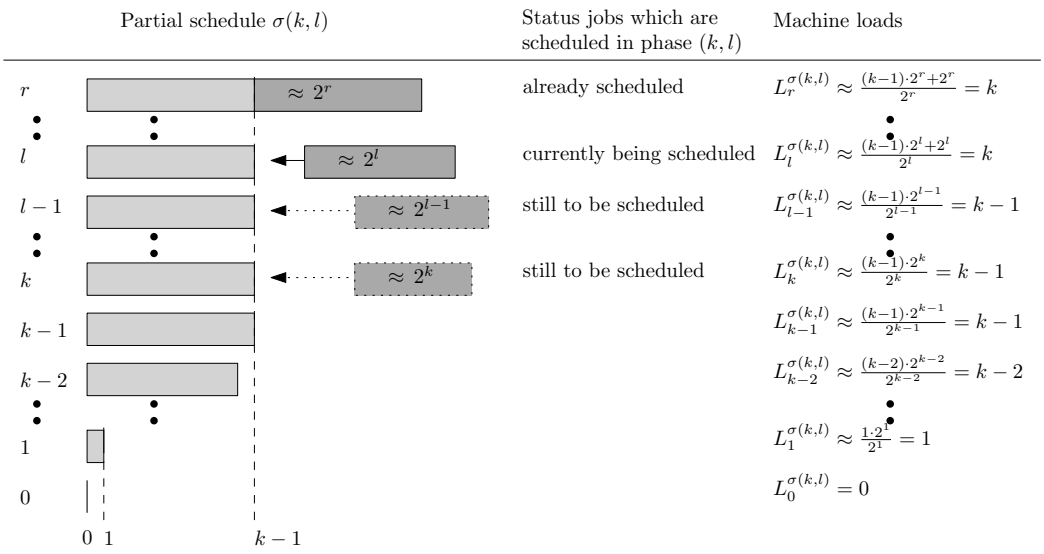

Figure 3.4: The partial schedule $\sigma(k,l)$ when the job in J_l are being assigned to the machines in M_l. Machine i in the figure above represents all machines in the class M_i.

Figure 3.4 visualizes the partial schedule $\sigma(k,l)$. Machine i with speed $s_i = 2^i$ is a representative for all machines in class M_i. With $L_i^{\sigma(k,l)}$ I refer to the current load of machine i after iteration (k,l). Similarly, $L_i^{\sigma(k,k)}$ denotes the load of machine i at the end of iteration (k,k), i.e., in the partial schedule $\sigma(k,k)$. In phase (k,l), $r!/l!$ jobs of size roughly 2^l are being assigned to the $r!/l!$ machines in M_l. All machines in $M_{l'}$ for $l' > l$ just received a job of roughly size $2^{l'}$. All machines in $M_{l'}$ for $l' \in \{k,\dots,l-1\}$ will still receive a single job of size roughly $2^{l'}$ during iteration k of the outer loop. Figure 3.4 follows from the observations. First, I validate the claims for the first iteration $(1,r)$. As only $r!/r! = 1$ job of class J_r has to be scheduled and since all machines are still empty, the job will be scheduled on the fastest machine which is the single machine in M_r. Hence, both claims hold true for the first iteration. Now consider an arbitrary iteration (k,l) and assume both claims hold true for all previous iterations. Consider a job $j \in J_l$ which needs to be assigned to a machine during iteration (k,l). The job j will then always be assigned to an unused machine $i \in M_l$. For this note that the previous iteration was either $(k,l+1)$ or $(k-1,k-1)$. Each of the following arguments is in particular based on the second claim.

- Any unused machine $i \in M_l$ carries $k-1$ jobs of class J_l. Consequently,

$$L_i^{\sigma(k,l)} + \frac{p_j}{s_i} < \frac{k\cdot(2^l + 2^{r+1}/\phi)}{2^l} = k + \frac{k}{\phi}\cdot 2^{r+1-l} \leq k + \frac{l}{2^{2r}}\cdot 2^{r+1-l} \leq k + \frac{1}{2^r}\,,$$

  where it is used that $k \leq l$, $\phi \geq 2^{2r}$, and $l/2^l \leq 1/2$ for all integers $l \geq 1$.

- Any machine h which is either used (in that case let $l' = l$) or in class $M_{l'}$ for some

$l' \in \{l+1, \dots, r\}$ carries k jobs of class $J_{l'}$ and thus

$$L_h^{\sigma(k,l)} + \frac{p_j}{s_h} \geq \frac{k \cdot 2^{l'} + 2^l}{2^{l'}} = k + 2^{l-l'} > k + \frac{1}{2^r} > L_i^{\sigma(k,l)} + \frac{p_j}{s_i}\,.$$

- Any machine $h \in M_{l'}$ for some $l' \in \{1, \dots, l-1\}$ carries $\min\{k-1, l'\}$ jobs of class $J_{l'}$ and thus

$$\begin{aligned} L_h^{\sigma(k,l)} + \frac{p_j}{s_h} &\geq \frac{\min\{k-1, l'\} \cdot 2^{l'} + 2^l}{2^{l'}} = \min\{k-1, l'\} + 2^{l-l'} \\ &\geq (k - \max\{k-l', 1\}) + 2^{\max\{k-l', 1\}} \geq k+1 > L_i^{\sigma(k,l)} + \frac{p_j}{s_i}\,, \end{aligned}$$

  where the second inequality follows from $l \geq \max\{k, l'+1\}$ and the third inequality follows from $2^i - i \geq 1$ for all positive integers i.

With this complete case analysis it has been shown that job j will be assigned to an unused machine $i \in M_l$. In conclusion, during iteration (k,l), each of the $r!/l!$ jobs to be assigned, will be assigned to an unused machine in M_l. Note that $|M_l| = r!/l!$, and hence for each job there always exists such an unused machine. The first claim and the second claim follow immediately. □

Lemma 3.2.23. *Schedule $\sigma(\mathcal{I})$ is lexjump optimal.*

Proof. It follows from Lemma 3.2.20 that the load of any machine $i \in M_l$ can be bounded by

$$l \leq L_i^{\sigma(\mathcal{I})} \leq l + l \cdot \frac{2^{r+1}}{2^l \phi} \leq l + \frac{l}{2^l} \cdot \frac{2^{r+1}}{2^{2r}} \leq l + \frac{1}{2} \cdot \frac{2^{r+1}}{2^{2r}} < l + 1\,.$$

If a job j currently scheduled on machine $i \in M_l$, would jump to another machine $i' \in M_{l'}$, then

$$L_{i'}^{\sigma(\mathcal{I})} + \frac{p_j}{s_{i'}} \geq l' + \frac{2^l}{2^{l'}} = l - (l - l') + 2^{l-l'} \geq l + 1 > L_i^{\sigma(\mathcal{I})}\,,$$

where the last inequality follows from $2^k - k \geq 1$ for all integers k. Thus, any job would be worse off by jumping to another machine, and hence schedule $\sigma(\mathcal{I})$ is lexjump optimal. □

Finally, Theorem 3.2.19 is proven.

Proof of Theorem 3.2.19. Consider the schedule $\sigma(\mathcal{I})$ which is returned by algorithm LB-LIST and which is a lexjump optimal schedule due to Lemma 3.2.23. By Lemma 3.2.20 the load of the single machine in M_r is at least r. Hence, $C_{\max}^{\sigma(\mathcal{I})} \geq r$. Now consider another schedule $\sigma_2(\mathcal{I})$ in which each machine in M_l processes a single job from job class J_{l+1}, $l = 0, \dots, r-1$. The single machine in M_r remains empty. Then, the load of any machine $i \in M_l$ with job j assigned to it is bounded as follows:

$$L_i^{\sigma_2(\mathcal{I})} = p_j / s_i \leq (2^{l+1} + 2^{r+1}/\phi)/2^l \leq 2 + 2^{r+1}/(2^{2r} \cdot 2^1) = 2 + 2^{-r} < 3\,.$$

Hence, $C_{\max}^{OPT(\mathcal{I})} \leq C_{\max}^{\sigma_2(\mathcal{I})} < 3$ and the theorem follows: $C_{\max}^{\sigma(\mathcal{I})} / C_{\max}^{OPT(\mathcal{I})} \geq r/3 = \Omega(r) = \Omega(\log \phi)$. □

3.2.4 List schedules on related parallel machines

With some more technicalities it can be shown that Theorem 3.2.5 also applies for the list scheduling algorithm at the cost of some higher constants. In [24], together with my co-authors I show that for β being an arbitrary positive real, for $\phi \geq 2$ and for any ϕ-smooth instance Φ with related parallel machines:

$$\mathop{\mathbf{Pr}}_{\mathcal{I}\sim\Phi}\left[\max_{\sigma(\mathcal{I})\in\mathrm{List}(\mathcal{I})} \frac{C_{\max}^{\sigma(\mathcal{I})}}{C_{\max}^{OPT(\mathcal{I})}} \geq \beta\right] \leq \left(\frac{32\phi}{2^{\beta/6}}\right)^{n/2}$$

and

$$\mathop{\mathbf{E}}_{\mathcal{I}\sim\Phi}\left[\max_{\sigma(\mathcal{I})\in\mathrm{List}(\mathcal{I})} \frac{C_{\max}^{\sigma(\mathcal{I})}}{C_{\max}^{OPT(\mathcal{I})}}\right] \leq 18\log\phi + 30 = O(\log\phi),$$

where $\mathrm{List}(\mathcal{I})$ denotes the set of all schedules which can be generated by the list scheduling algorithm on some instance $\mathcal{I}$.

Since the lower bound example constructed in Subsection 3.2.3 is generated by the use of the list scheduling algorithm, see line 3 of algorithm LB-LIST, the lower bound applies straightforwardly for the list scheduling algorithm as well. Thus, there is a class of ϕ-smooth instances Φ on related parallel machines such that, for any $\mathcal{I} \in \Phi$,

$$\max_{\sigma(\mathcal{I})\in\mathrm{List}(\mathcal{I})} \frac{C_{\max}^{\sigma(\mathcal{I})}}{C_{\max}^{OPT(\mathcal{I})})} = \Omega(\log\phi)\,.$$

3.3 Restricted parallel machines

In this section lower bound examples are provided showing that the worst-case performance guarantees for all variants of the restricted machines are robust against random noise. Even with large perturbations, that is for a constant ϕ, the worst-case lower bounds still apply.

3.3.1 Jump neighborhood on restricted related machines

Theorem 2.2.2 of the previous chapter shows that the makespan of a jump optimal schedule is at most a factor of $1/2 + \sqrt{m - 3/4}$ away from the optimal makespan on restricted identical parallel machines. On restricted related parallel machines, Theorem 2.2.1 showed that the makespan of a jump optimal schedule is not more than a factor of

$$1/2 + \sqrt{(m-1)\cdot(s_{\max}/s_{\min}) + 1/4}$$

away from the makespan of an optimal schedule. Additionally, both bounds are tight up to a constant factor. Here, I show that even on ϕ-smooth instances both bounds are tight up to a constant factor.

Theorem 3.3.1. *For every $\phi \geq 2$ there exists a class of ϕ-smooth instances Φ on restricted related parallel machines such that*

$$\mathop{\mathbf{E}}_{\mathcal{I}\sim\Phi}\left[\max_{\sigma(\mathcal{I})\in\mathrm{Jump}(\mathcal{I})} \frac{C_{\max}^{\sigma(\mathcal{I})}}{C_{\max}^{OPT(\mathcal{I})}}\right] = \Omega\left(\sqrt{m\cdot\frac{s_{\max}}{s_{\min}}}\right)\,.$$

Proof. It suffices to show the claim for $\phi = 2$ and $m \geq 3$. Without loss of generality it assumed that $s_{\min} = 1$. Let $s := s_{\max}/s_{\min} = s_{\max}$. Further, let $z > 2$ be an arbitrary integer, and

$$m' = m - 2 \geq 1\,, \quad k' = \sqrt{\frac{m'}{s}} \leq \sqrt{m'}\,, \quad \text{and } k = \lceil k' \rceil\,.$$

In the remainder assume that $\sqrt{m's} \geq 17$. Consider the following ϕ-smooth instance Φ. The set M of machines is partitioned into three classes M_1, M_2, and M_3 such that

$$|M_1| = 1\,, \quad |M_2| = k\,, \quad \text{and } |M_3| = m' - (k-1) > m' - k' \geq 0\,.$$

The machine in M_1 has speed 1, the machines in M_2 have speed

$$s' = \max\left\{1, s \cdot \frac{k'}{k}\right\} \in [1, s]\,,$$

and the machines in M_3 have speed s. Let the set J of jobs be partitioned into two subsets J_1 and J_2, consisting of

$$|J_1| = \left\lfloor 2zsk' \right\rfloor \quad \text{and} \quad |J_2| = \left\lceil 32zs \cdot (m' - k') \right\rceil \leq \left\lceil 32zs \cdot |M_3| \right\rceil$$

jobs whose processing requirements are independently and uniformly drawn from $[1/2, 1]$ and from $[0, 1/2]$ respectively, yielding a realization $\mathcal{I} \sim \Phi$. The jobs in J_1 are only allowed to be scheduled on the machines in $M_1 \cup M_2$, whereas the jobs in J_2 are allowed to be scheduled on any machine.

First, a schedule $\sigma_2(\mathcal{I})$ is constructed to bound the optimal makespan: use the list scheduling to schedule all jobs in J_1 on the machines in M_2, and all jobs in J_2 on the machines in M_3. Figure 3.5 depicts schedule $\sigma_2(\mathcal{I})$. It follows that for all machines $i \in M_2$

$$L_i^{\sigma_2(\mathcal{I})} \leq \frac{\frac{\sum_{j \in J_1} p_j}{|M_2|} + \max\limits_{j \in J_1} p_j}{s'} \leq \frac{\frac{|J_1| \cdot 1}{|M_2|} + 1}{s'} \leq \frac{\frac{2zsk'}{k} + 1}{s'} \leq \frac{\frac{2zsk'}{k}}{s \cdot \frac{k'}{k}} + \frac{1}{1} = 2z + 1\,.$$

Similarly, for all machines $i \in M_3$

$$L_i^{\sigma_2(\mathcal{I})} \leq \frac{\frac{\sum_{j \in J_2} p_j}{|M_3|} + \max\limits_{j \in J_2} p_j}{s} \leq \frac{\frac{|J_2| \cdot \frac{1}{2}}{|M_3|} + \frac{1}{2}}{s} \leq \frac{\frac{32zs \cdot |M_3|}{2 \cdot |M_3|} + 1}{s} \leq 16z + 1\,.$$

Hence, $C_{\max}^{OPT(\mathcal{I})} \leq C_{\max}^{\sigma_2(\mathcal{I})} \leq 17z$. Before proceeding, with constructing a 'bad' jump optimal schedule $\sigma(\mathcal{I})$, first observe that

$$s' \leq 2s \cdot k'/k \tag{3.7}$$

due to $1 \leq (\sqrt{m'} + 1)/k \leq 2\sqrt{m's}/k = 2s \cdot k'/k$.

I construct a jump optimal schedule $\sigma(\mathcal{I})$ for a realization $\mathcal{I}$ of the ϕ-smooth instance Φ such that the corresponding makespan exceeds zsk' with high probability: schedule all jobs in J_1 on the single machine in M_1. Then, $zsk' - 1 \leq L_1^{\sigma(\mathcal{I})} \leq 2zsk'$. Next, start assigning jobs from J_2 to the machines in M_2 with the list scheduling and an arbitrary job permutation, until

(a) either J_2 becomes empty, or until

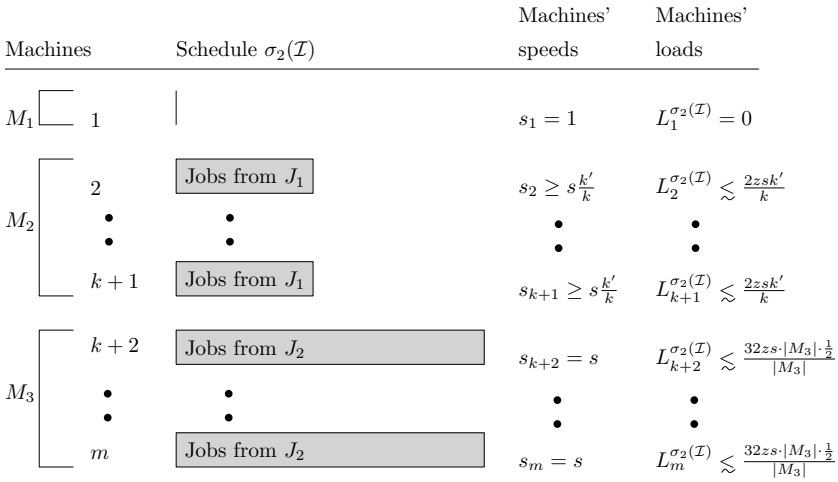

Figure 3.5: Schedule $\sigma_2(\mathcal{I})$

(b) $L_i^{\sigma(\mathcal{I})} \in \left[L_1^{\sigma(\mathcal{I})} - \frac{1}{2s'}, L_1^{\sigma(\mathcal{I})}\right)$ for all $i \in M_2$. Any remaining unscheduled jobs in J_2 are assigned to the machines in M_3 using list scheduling.

Let $Q = \sum_{j \in J_2} p_j$ and let $\mathcal{E}$ denote the event that $Q > 4z(sk')^2$. If $\mathcal{E}$ occurs, then

$$\sum_{i \in M_2} s' \cdot L_1^{\sigma(\mathcal{I})} \leq |M_2| \cdot \left(2s \cdot \frac{k'}{k}\right) \cdot 2zsk' = 4z(sk')^2 < Q$$

due to inequality (3.7), i. e., the algorithm will end up in case (b) as $p_j \leq 1/2$ for any job $j \in J_2$. This shows that no machine $i \in M_2$ is critical. Using the same argument as for the analysis of $\sigma_2(\mathcal{I})$, one can show that the load of any machine $i \in M_3$ is bounded from above by $16z + 1 < 17z - 1 \leq z \cdot \sqrt{m' \cdot s} - 1 = zsk' - 1 \leq L_1^{\sigma(\mathcal{I})}$, i.e., the machine in M_1 is the unique critical machine. As each job on this machine has processing requirement at least $1/2$ and due to the property of the loads of the machines in M_2 in case (b), schedule $\sigma(\mathcal{I})$ is jump optimal and $C_{\max}^{\sigma(\mathcal{I})} = L_1^{\sigma(\mathcal{I})} \geq zsk' - 1$.

It remains to determine the probability $\mathbf{Pr}[\mathcal{E}]$. For this, note that

$$\mathbf{E}[Q] = \frac{|J_2|}{4} \geq 8zs \cdot (m' - k') = 8zsm' \cdot \left(1 - \frac{k'}{m'}\right) = 8zsm' \cdot \left(1 - \frac{1}{\sqrt{m's}}\right) > 6zsm'$$

as $\sqrt{m's} \geq 17$ by our initial assumption. On the other hand, $4z(sk')^2 = 4zsm'$. Applying Hoeffding's Inequality [67], it holds that

$$\begin{aligned}\mathbf{Pr}\left[\bar{\mathcal{E}}\right] = \mathbf{Pr}\left[Q \leq 4zsm'\right] &\leq \mathbf{Pr}\left[Q - \mathbf{E}[Q] \leq -2zsm'\right] \\ &\leq \exp\left(-\frac{2 \cdot (2zsm')^2}{|J_2| \cdot \left(\frac{1}{2}\right)^2}\right) \leq \exp\left(-\frac{32z^2s^2(m')^2}{32zsm' + 1}\right),\end{aligned}$$

which becomes arbitrarily close to 0 with respect to z. Hence, for sufficiently large integers z

$$\begin{aligned}\mathop{\mathbf{E}}_{\mathcal{I}\sim\Phi}\left[\max_{\sigma(\mathcal{I})\in\mathrm{Jump}(\mathcal{I})}\frac{C_{\max}^{\sigma(\mathcal{I})}}{C_{\max}^{OPT(\mathcal{I})}}\right] &\geq \mathop{\mathbf{E}}_{\mathcal{I}\sim\Phi}\left[\max_{\sigma(\mathcal{I})\in\mathrm{Jump}(\mathcal{I})}\frac{C_{\max}^{\sigma(\mathcal{I})}}{C_{\max}^{OPT(\mathcal{I})}}\middle|\,\mathcal{E}\right]\cdot\mathop{\mathbf{Pr}}_{\mathcal{I}\sim\Phi}[\mathcal{E}]\\ &\geq \frac{zsk'-1}{17z}\cdot\frac{17}{18}\geq\frac{\sqrt{(m-2)\cdot\frac{s_{\max}}{s_{\min}}}-\frac{1}{z}}{18}\,.\end{aligned}$$

□

Corollary 3.3.2. *For every $\phi \geq 2$ there exists a class of ϕ-smooth instances Φ on restricted identical machines such that*

$$\mathop{\mathbf{E}}_{\mathcal{I}\sim\Phi}\left[\max_{\sigma(\mathcal{I})\in\mathrm{Jump}(\mathcal{I})}\frac{C_{\max}^{\sigma(\mathcal{I})}}{C_{\max}^{OPT(\mathcal{I})}}\right]=\Omega(\sqrt{m})\,.$$

Remark 3.3.3. In the proof of Theorem 3.3.1 I introduce an arbitrary integer z. I argue that there exists a sufficiently large value for z such that the desired result follows. Choosing an even larger value for z implies that the results above not only hold in expectation but also with high probability.

3.3.2 Lexjump neighborhood on restricted identical machines

In this subsection I show that there exist instances with $\phi \geq 8$ such that the smoothed performance guarantee for lexjump optimal schedules in the restricted setting is in the same order as the worst case performance guarantee.

Theorem 3.3.4. *For every $\phi \geq 8$ there exists a class of ϕ-smooth instances Φ on restricted identical parallel machines such that*

$$\mathop{\mathbf{E}}_{\mathcal{I}\sim\Phi}\left[\max_{\sigma(\mathcal{I})\in\mathrm{Lex}(\mathcal{I})}\frac{C_{\max}^{\sigma(\mathcal{I})}}{C_{\max}^{OPT(\mathcal{I})}}\right]=\Omega\left(\frac{\log m}{\log\log m}\right)\,.$$

First, let me introduce the ϕ-smooth instance Φ for $\phi \geq 8$. Given an integer $k \geq 68$, consider the following recurrence formula:

$$a_0=k^2\,,\qquad a_1=k^3\,,\quad\text{and}\quad a_h=\left\lceil\left(\frac{a_{h-1}}{a_{h-2}}-\frac{7}{15}\right)\cdot a_{h-1}\right\rceil\ \text{for } h\geq 2\,.$$

Starting with $a_1/a_0 = k$, the fraction a_h/a_{h-1} decreases with increasing index h until it is less than or equal to 1. Let z_k be the smallest integer h such that $a_h/a_{h-1} \leq 1$. Also, $a_0, a_1, \ldots, a_{z_k-1}$ is a strictly increasing sequence. The number z_k will be bounded from above later in the analysis.

Consider z_k job classes $J_1, \ldots, J_{z_k}$ and as many machine classes $M_1, \ldots, M_{z_k}$. Each machine class M_h contains $m_h = a_{h-1}$ machines with speed 1. Each job class J_h consists of two subclasses J_h^A and J_h^B of size a_h and of size $b_h = 17m_h$, respectively. The jobs in class J_h^A are called type A jobs, have processing requirements independently and uniformly distributed in $[7/8, 1]$, and can be processed on machines in $M_h \cup M_{h+1}$. As a convention let $M_{z_k+1} = \emptyset$.

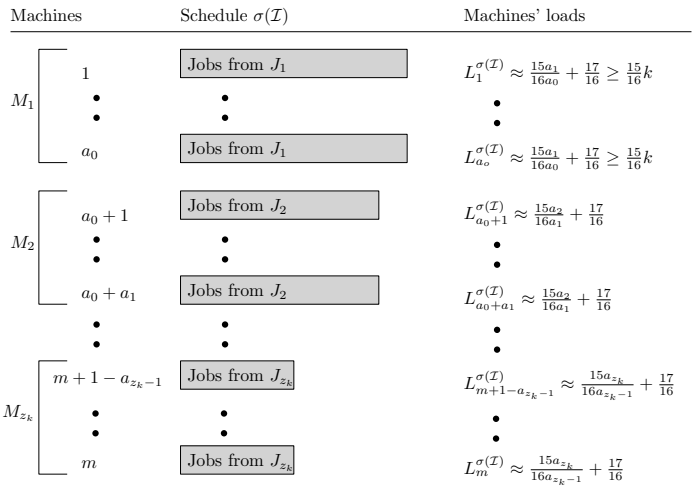

Figure 3.6: Schedule $\sigma(\mathcal{I})$ illustrating Theorem 3.3.4

Jobs in class J_h^B are called type B jobs, have processing requirements independently and uniformly distributed in $[0, 1/8]$, and can only be processed on machines in M_h.

The schedule $\sigma(\mathcal{I})$ for a realization $\mathcal{I} \in \Phi$ is obtained by scheduling the jobs in J_h on the machines in M_h using LPT (longest processing time) scheduling, i.e., list scheduling with a list in which the jobs are ordered according to non-increasing processing requirements. Note that the LPT algorithm first schedules all type A jobs and then all type B jobs. Schedule $\sigma(\mathcal{I})$ is visualized in Figure 3.6. Machine h represents all machines in class M_h.

I show that schedule $\sigma(\mathcal{I})$ is lexjump optimal with high probability. To be more specific, I show lexjump optimality when the values $Q_h^A = \sum_{j \in J_h^A} p_j$ and $Q_h^B = \sum_{j \in J_h^B} p_j$ are close to their expectations. Let $\mathcal{E}_h^A$ and $\mathcal{E}_h^B$ denote the events that

$$\left| Q_h^A - \mathbf{E}\left[Q_h^A\right] \right| \leq \frac{m_h}{16} \quad \text{and} \quad \left| Q_h^B - \mathbf{E}\left[Q_h^B\right] \right| \leq \frac{m_h}{32}, \quad \text{respectively}.$$

Moreover, let $\mathcal{E}$ denote the event that the events $\mathcal{E}_h^A$ and $\mathcal{E}_h^B$ are simultaneously true for all $h = 1, \ldots, z_k$. Also, let $\bar{\mathcal{E}}_h^A$, $\bar{\mathcal{E}}_h^B$, and $\bar{\mathcal{E}}$ refer to the complement of $\mathcal{E}_h^A$, $\mathcal{E}_h^B$, and $\mathcal{E}$.

First, I analyze the sequence $a_0, a_1, \ldots, a_{z_k}$ to obtain bounds for the number z_k of machine and job classes and for the number m of machines.

Lemma 3.3.5. *For any $h = 1, \ldots, z_k$ the following inequality holds:*

$$\frac{a_h}{a_{h-1}} \leq k - (h-1) \cdot \frac{2}{5}.$$

Proof. The claim is true for $h = 1$. By definition of a_h,

$$\frac{a_h}{a_{h-1}} \leq \frac{\left(\frac{a_{h-1}}{a_{h-2}} - \frac{7}{15}\right) \cdot a_{h-1} + 1}{a_{h-1}} \leq \frac{a_{h-1}}{a_{h-2}} - \frac{6}{15} = \frac{a_{h-1}}{a_{h-2}} - \frac{2}{5}$$

for any $h = 2, \ldots, z_k$ as $a_{h-1} \geq a_0 = k^2 \geq 15$. The claim follows by induction. □

Now the number z_k of job classes can be bounded.

Corollary 3.3.6. *The number z_k of machine and job classes is bounded by $5k/2$.*

Proof. Applying Lemma 3.3.5 for $h = z_k - 1$ it follows that

$$1 < \frac{a_{z_k-1}}{a_{z_k-2}} \leq k - (z_k - 2) \cdot \frac{2}{5}\,.$$

Hence,

$$z_k < (k-1) \cdot \frac{5}{2} + 2 < \frac{5k}{2}\,.$$

□

Lemma 3.3.7. *The number m of machines is bounded by $\Gamma(k'+3)$ where Γ denotes the gamma function and where $k' = \lceil 5k/2 \rceil$.*

Proof. By induction it will be shown that

$$a_h \leq k^2 \cdot \left(\frac{2}{5}\right)^h \cdot \frac{k'!}{(k'-h)!}$$

for any $h = 0, \ldots, z_k - 1$. Note that $z_k \leq 5k/2 \leq k'$ due to Corollary 3.3.6. For $h = 0$ the claim holds since $a_0 = k^2$. For $h \geq 1$ apply Lemma 3.3.5 to get

$$\frac{a_h}{a_{h-1}} \leq k - (h-1) \cdot \frac{2}{5} \leq \frac{2}{5} \cdot (k' - (h-1))\,.$$

The induction hypothesis for a_{h-1} yields

$$a_h \leq \frac{2}{5} \cdot (k' - (h-1)) \cdot k^2 \cdot \left(\frac{2}{5}\right)^{h-1} \cdot \frac{k'!}{(k'-(h-1))!} = k^2 \cdot \left(\frac{2}{5}\right)^h \cdot \frac{k'!}{(k'-h)!}\,.$$

Recalling $m_h = a_{h-1}$ the number m of machines can be bounded by

$$\frac{m}{k^2} = \sum_{h=1}^{z_k} \frac{m_h}{k^2} = \sum_{h=0}^{z_k-1} \frac{a_h}{k^2} \leq \sum_{h=0}^{z_k-1} \frac{k'!}{(k'-h)!} \leq k'! \cdot e\,.$$

Hence, $m \leq e \cdot k^2 \cdot k'! \leq (k'+2)! = \Gamma(k'+3)$. □

Lemma 3.3.8. *Event $\bar{\mathcal{E}}$ occurs with probability at most $10k \cdot e^{-k/2}$.*

Proof. The aim is to bound the probability for the events $\bar{\mathcal{E}}_h^A$ and $\bar{\mathcal{E}}_h^B$ to occur. Recalling $m_h = a_{h-1} \geq a_0 = k^2$, $a_h \leq k \cdot a_{h-1}$ (see Lemma 3.3.5), $b_h = 17m_h$, and $k \geq 68$ it follows that

$$\begin{aligned}\mathbf{Pr}\left[\bar{\mathcal{E}}_h^A\right] &= \mathbf{Pr}\left[\left|Q_h^A - \mathbf{E}[Q_h^A]\right| > \frac{m_h}{16}\right] \leq 2\exp\left(-\frac{2\left(\frac{m_h}{16}\right)^2}{a_h \cdot \left(\frac{1}{8}\right)^2}\right) \\ &= 2\exp\left(-\frac{a_{h-1}}{a_h} \cdot \frac{a_{h-1}}{2}\right) \leq 2\exp\left(-\frac{a_{h-1}}{2k}\right) \leq 2\exp\left(-\frac{k}{2}\right)\end{aligned}$$

and

$$\begin{aligned}\mathbf{Pr}\left[\bar{\mathcal{E}}_h^B\right] &= \mathbf{Pr}\left[\left|Q_h^B - \mathbf{E}[Q_h^B]\right| > \frac{m_h}{32}\right] \leq 2\exp\left(-\frac{2\left(\frac{m_h}{32}\right)^2}{b_h \cdot \left(\frac{1}{8}\right)^2}\right) \\ &= 2\exp\left(-\frac{m_h}{17m_h} \cdot \frac{a_{h-1}}{8}\right) \leq 2\exp\left(-\frac{k^2}{136}\right) \leq 2\exp\left(-\frac{k}{2}\right).\end{aligned}$$

Each of the first inequalities stems from Hoeffding's bound [67]. A union bound yields

$$\mathbf{Pr}\left[\bar{\mathcal{E}}\right] = \mathbf{Pr}\left[\bigcup_{h=1}^{z_k}\left(\bar{\mathcal{E}}_h^A \cup \bar{\mathcal{E}}_h^B\right)\right] \leq 2z_k \cdot 2\exp\left(-\frac{k}{2}\right) \leq 10k \cdot \exp\left(-\frac{k}{2}\right)$$

due to Corollary 3.3.6. □

As event $\mathcal{E}$ occurs with high probability and as

$$\mathop{\mathbf{E}}_{\mathcal{I}\sim\Phi}\left[\max_{\sigma(\mathcal{I})\in\mathrm{Lex}(\mathcal{I})}\frac{C_{\max}^{\sigma(\mathcal{I})}}{C_{\max}^{OPT(\mathcal{I})}}\right] \geq \mathop{\mathbf{E}}_{\mathcal{I}\sim\Phi}\left[\max_{\sigma(\mathcal{I})\in\mathrm{Lex}(\mathcal{I})}\frac{C_{\max}^{\sigma(\mathcal{I})}}{C_{\max}^{OPT(\mathcal{I})}}\middle|\,\mathcal{E}\right]\cdot\mathop{\mathbf{Pr}}_{\mathcal{I}\sim\Phi}\left[\mathcal{E}\right],$$

to prove Theorem 3.3.4 it suffices to bound the expected value conditioned on event $\mathcal{E}$ by $\Omega\left(\frac{\log m}{\log\log m}\right)$. Therefore, in the remainder of this subsection, it is assumed that event $\mathcal{E}$ is true.

Lemma 3.3.9. *The loads of the machines within the same class differ only slightly. In particular,* $|L_i^{\sigma(\mathcal{I})} - L_{i'}^{\sigma(\mathcal{I})}| \leq 1/8$ *for any machines* $i, i' \in M_h$.

Proof. Suppose to the contrary that there exist two machines $i, i' \in M_h$ such that $L_i^{\sigma(\mathcal{I})} - L_{i'}^{\sigma(\mathcal{I})} > 1/8$. Recall that according to the LPT rule all type A jobs will be assigned to the machines before the type B jobs are assigned. After all type A jobs have been assigned to the machines in M_h, the difference in load between any two machines in M_h is at most 1 since $p_j \leq 1$ for all jobs j.

Since the processing time of all type B jobs is bounded by $1/8$, $L_i^{\sigma(\mathcal{I})} - L_{i'}^{\sigma(\mathcal{I})} > 1/8$ implies that no type B job is assigned to machine i nor to any machine that has load at least $L_i^{\sigma(\mathcal{I})}$. Hence, all type B jobs are assigned to the machines that have load less than $L_i^{\sigma(\mathcal{I})}$. Note that there are at most $m_h - 1$ such machines.

As the difference in load between machine i and any other machine in M_h is at most 1, the total amount of processing requirements of type B jobs in class M_h is bounded by $Q_h^B \leq (m_h - 1) \cdot 1 < 17m_h/16 - m_h/32 = \mathbf{E}[Q_h^B] - m_h/32$ contradicting the assumption that event $\mathcal{E}_h^B$ holds. □

Lemma 3.3.10. *For any machine* $i \in M_h$ *the inequality*

$$\left|L_i^{\sigma(\mathcal{I})} - \frac{1}{m_h}\left(\mathbf{E}\left[Q_h^A\right] + \mathbf{E}\left[Q_h^B\right]\right)\right| \leq \frac{7}{32}$$

holds, i.e., the load of machine i *is close to the expected average machine load in class* M_h.

Proof. Applying the triangle inequality yields

$$\left| L_i^{\sigma(\mathcal{I})} - \frac{\mathbf{E}\left[Q_h^A\right] + \mathbf{E}\left[Q_h^B\right]}{m_h} \right| \leq \left| L_i^{\sigma(\mathcal{I})} - \frac{Q_h^A + Q_h^B}{m_h} \right| + \frac{\left| Q_h^A - \mathbf{E}\left[Q_h^A\right] \right|}{m_h} + \frac{\left| Q_h^B - \mathbf{E}\left[Q_h^B\right] \right|}{m_h}$$
$$\leq \left| L_i^{\sigma(\mathcal{I})} - \frac{\sum_{i' \in M_h} L_{i'}^{\sigma(\mathcal{I})}}{|M_h|} \right| + \frac{1}{16} + \frac{1}{32} \leq \frac{7}{32}\,,$$

where the second inequality holds since $\mathcal{E}_h^A$ and $\mathcal{E}_h^B$ are true. The last inequality is due to Lemma 3.3.9. □

Lemma 3.3.11. *Schedule $\sigma(\mathcal{I})$ is lexjump optimal.*

Proof. It needs to be shown that $L_{i'}^{\sigma(\mathcal{I})} + p_j \geq L_i^{\sigma(\mathcal{I})}$ holds for any machine $i \in M_h$, any job $j \in J_i^{\sigma(\mathcal{I})}$, and any machine $i' \in \mathcal{M}_j$. Let $i \in M_h$ be an arbitrary machine. First, consider the last job j that has been assigned to i. Then, $L_{i'}^{\sigma(\mathcal{I})} + p_j \geq L_i^{\sigma(\mathcal{I})}$ for any machine $i' \in M_h$ as this job was assigned to machine i by list scheduling. Furthermore, job j is a smallest job on machine i due to the LPT rule. Hence, $L_{i'}^{\sigma(\mathcal{I})} + p_{j'} \geq L_i^{\sigma(\mathcal{I})}$ for any machine $i' \in M_h$ and any job $j' \in J_i^{\sigma(\mathcal{I})}$ assigned to machine i.

For type B jobs on machine i the set of allowed machines equals M_h. It just remains to show that $L_{i'}^{\sigma(\mathcal{I})} + p_j \geq L_i^{\sigma(\mathcal{I})}$ for any machine $i' \in M_{h+1}$ and any type A job $j \in J_i^{\sigma(\mathcal{I})}$. Recalling $a_h = \lceil (a_{h-1}/a_{h-2}) - 7/15) \cdot a_{h-1} \rceil$ for $h \geq 2$, $m_h = a_{h-1}$, and $b_h/m_h = 17$ finds

$$\frac{\mathbf{E}\left[Q_{h+1}^A\right] + \mathbf{E}\left[Q_{h+1}^B\right]}{m_{h+1}} = \frac{\frac{15}{16}a_{h+1} + \frac{1}{16}b_{h+1}}{m_{h+1}} = \frac{15}{16} \cdot \frac{a_{h+1}}{a_h} + \frac{1}{16} \cdot \frac{b_{h+1}}{m_{h+1}}$$
$$\geq \frac{15}{16} \cdot \left(\frac{a_h}{a_{h-1}} - \frac{7}{15} \right) + \frac{1}{16} \cdot \frac{b_h}{m_h} = \frac{\mathbf{E}\left[Q_h^A\right] + \mathbf{E}\left[Q_h^B\right]}{m_h} - \frac{7}{16}$$

for any $h = 1, \ldots, z_k - 1$. This implies

$$L_{i'}^{\sigma(\mathcal{I})} + p_j \geq \frac{\mathbf{E}\left[Q_{h+1}^A\right] + \mathbf{E}\left[Q_{h+1}^B\right]}{m_{h+1}} - \frac{7}{32} + \frac{7}{8}$$
$$\geq \frac{\mathbf{E}\left[Q_h^A\right] + \mathbf{E}\left[Q_h^B\right]}{m_h} - \frac{7}{16} + \frac{21}{32} = \frac{\mathbf{E}\left[Q_h^A\right] + \mathbf{E}\left[Q_h^B\right]}{m_h} + \frac{7}{32} \geq L_i^{\sigma(\mathcal{I})}\,,$$

where the first and the last inequality are due to Lemma 3.3.10. □

Finally, Theorem 3.3.4 can be shown.

Proof of Theorem 3.3.4. As mentioned before, due to Lemma 3.3.8 it suffices to bound the expected value conditioned on event $\mathcal{E}$. If event $\mathcal{E}$ holds, then schedule $\sigma(\mathcal{I})$ is lexjump optimal (see Lemma 3.3.11), i.e., $\sigma \in \mathrm{Lex}(\mathcal{I})$, and has makespan

$$C_{\max}^{\sigma(\mathcal{I})} \geq \max_{i \in M_1} L_i^{\sigma(\mathcal{I})} \geq \frac{Q_1^A + Q_1^B}{m_1} \geq \frac{\mathbf{E}[Q_1^A] + \mathbf{E}[Q_1^B]}{m_1} - \frac{\frac{m_1}{16} + \frac{m_1}{32}}{m_1} = \frac{\frac{15}{16}k^3 + \frac{1}{16} \cdot 17k^2}{k^2} - \frac{3}{32} \geq \frac{15}{16}k\,,$$

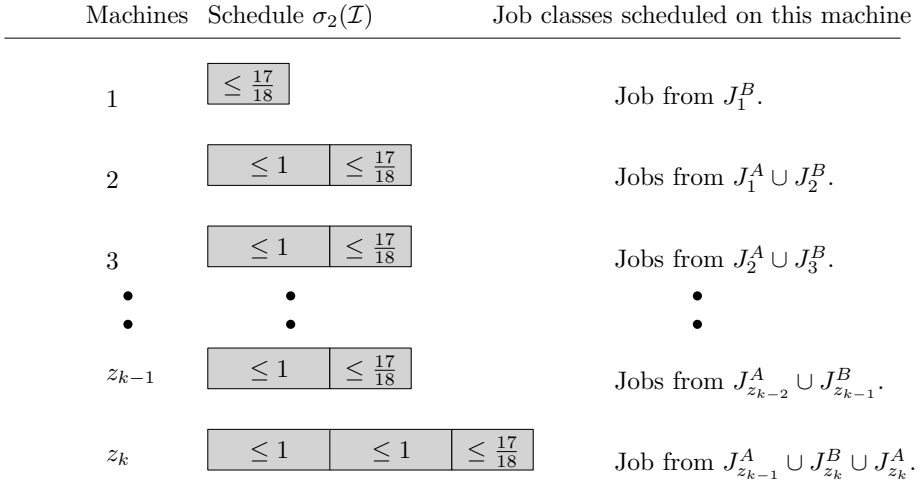

Figure 3.7: Schedule $\sigma_2(\mathcal{I})$ where each machine i is a representative for all machines in the clas M_i.

where the third inequality is due to the occurrence of $\mathcal{E}_1^A$ and $\mathcal{E}_1^B$. Now consider the following schedule $\sigma_2(\mathcal{I})$:

- For $h = 1, \ldots, z_k - 1$ spread the jobs of class J_h^A evenly among the machines in class M_{h+1}. As $|J_h^A| = a_h = m_{h+1} = |M_{h+1}|$, each machine is assigned exactly one type A job.
- Spread the jobs of class $J_{z_k}^A$ evenly among the machines in class M_{z_k}. As $|J_{z_k}^A| = a_{z_k} \leq a_{z_k - 1} = m_{z_k} = |M_{z_k}|$, each machine is assigned at most one type A job.
- For $h = 1, \ldots, z_k$ spread the jobs of class J_h^B evenly among the machines in class M_h. As $|J_h^B| = 17 m_h = 17 \cdot |M_h|$, each machine is assigned exactly 17 type B jobs.

Note that with 'evenly' I refer to the number of jobs on each machine and not to the load. Figure 3.7 shows schedule $\sigma_2(\mathcal{I})$ where each machine h is a representative for all machines in class M_h.

As each machine contains at most 2 type A jobs and 17 type B jobs, the makespan of schedule $\sigma_2(\mathcal{I})$, and hence $C_{\max}^{OPT(\mathcal{I})}$ is bounded by $2 \cdot 1 + 17 \cdot 1/8 \leq 5$. This implies $C_{\max}^{\sigma(\mathcal{I})} / C_{\max}^{OPT(\mathcal{I})} \geq 3k/16 = \Omega(\Gamma^{-1}(m))$ due to Lemma 3.3.7. Hence,

$$\mathop{\mathbf{E}}_{\mathcal{I} \sim \Phi} \left[\max_{\sigma(\mathcal{I}) \in \mathrm{Lex}(\mathcal{I})} \frac{C_{\max}^{\sigma(\mathcal{I})}}{C_{\max}^{OPT(\mathcal{I})}} \middle| \mathcal{E} \right] = \Omega\left(\frac{\log m}{\log \log m} \right).$$

□

Remark 3.3.12. Lemma 3.3.8 established that $\mathcal{E}$ occurs with high probability. Hence, if one chooses k suitably large, the stated results not only hold in expectation, but also with high probability.

Remark 3.3.13. The upper bound on the performance guarantee for lexjump optimal schedules on restricted related parallel machines is $O\left(\frac{\log S}{\log \log S}\right)$, where $S = \sum_i s_i/s_m$, see Theorem 2.3.3. As for identical machines $S = m$, i.e., each machine has speed 1, the upper bound matches the lower bound of Theorem 3.3.4 up to a constant factor and smoothing does also not improve the performance guarantee for the worst lexjump optimal schedules on restricted identical parallel machines.

3.4 Smoothing in routing games

A lexjump optimal schedule can be seen as a Nash equilibrium when the jobs are interpret as agents who want to be scheduled on a minimal loaded machine. Therefore, the performance guarantee of the lexjump neighborhood is equivalent to what is known in game theory as the price of anarchy. Subsection 3.2.2 then showed that the smoothed price of anarchy is a constant depending on the logarithm of the smoothing parameter ϕ, whereas the original price of anarchy was growing in the number of machines m. That is, under smoothing, the quality of Nash equilibria can be much better than the worst case approximation ratios suggest. In this section, I build upon this observation and analyze the quality of Nash equilibria in the presence of some noise. The focus is on one more popular problem from algorithmic game theory, namely routing games. The final result which is presented in Subsection 3.4.3 is only a preliminary result indicating that the smoothed price of anarchy might present a more positive and realistic insight on the quality of Nash equilibria than the traditional price of anarchy does.

3.4.1 Game theory and routing games

Let me start with introducing some notions from game theory, show the link to Subsection 3.2.2, and introduce routing games.

A *game* consists of a set of players or agents, a set of moves (or strategies) available to those agents, and a specification of payoffs received by the agents for each combination of strategies. I consider non-cooperative games wherein each agent only cares about optimizing his own payoff. A *state of a game* is a situation in which each agent has chosen one of his admissible strategies. In game theory, a *Nash Equilibrium* (NE) is a state of a game involving two or more agents, in which no agent can improve his payoff by changing his strategy unilaterally. If each agent has chosen a strategy and no agent can benefit by changing his or her strategy while the other agents keep theirs unchanged, then the current set of strategy choices and corresponding payoffs constitute a Nash equilibrium. The *Price of Anarchy* (PoA), introduced by Koutsoupias and Papadimitriou [75], measures how the efficiency of a system degrades due to selfish behavior of its agents. That is, the price of anarchy is the worst case ratio of the quality of a Nash equilibrium over the quality of the best possible solution (which need not be a NE).

Consider the problem studied in Subsection 3.2.2: I studied the performance guarantee of the lexjump neighborhood on related parallel machines with respect to minimizing the makespan. Interpreting jobs as agents who want to minimize the load they experience, i. e., the load of the machine on which they are scheduled. The resulting solution will be a lexjump

optimal schedule as well as a Nash equilibrium. The price of anarchy in this case is the ratio of the worst Nash equilibrium possible over the best possible solution, i. e., the price of anarchy equals the performance guarantee of the lexjump neighborhood. For this scheduling problem, Czumaj and Vöcking [38] showed that the price of anarchy can be quite bad and is growing in the number of machines m, namely $PoA = \Theta(\log m / \log\log m)$. The matching lower bound though is quite fragile. That is, if a few agents deviate from the strategy they follow in the lower bound example, which might be due to small changes in their payoff rule, then the whole structure of the worst case example breaks down and the approximation ratio drops. Indeed, I showed that the smoothed price of anarchy is $\Theta(\log \phi)$ where $1/\phi$ is a measure of the size of the perturbations. I conjecture that this is the case for many games.

Conjecture 3.4.1. *The price of anarchy of many games is fragile towards some agents deviating from their strategy as a result of slight perturbations in the input data.*

This conjecture implies that the quality of a Nash equilibrium observed in practice is often better than the price of anarchy suggests. To found the conjecture, I consider the smoothed price of anarchy of one more type of games, namely routing games also known as traffic assignment games.

Buriol et al. [25] introduce routing games in the following way. In *routing games*, traffic participants, here called *agents*, choose routes in a given road network so as to minimize their individual travel times. Each arc, i. e., road, has an associated latency function which expresses the flow-dependent delay that agents experience if they travel along that arc. That is, a latency function specifies the travel time along an arc and is a function depending on the number of agents which make use of that arc. The goal of every agent is to choose a path from its origin to its destination such that the total delay to travel along this route is minimized. The quality of a solution is measured by the total latency over all agents. Because the delay of each agent also depends on the choices made by the others, this problem can also naturally be interpreted as a strategic game in which agents (players) compute for resources (routes) and every agent acts *selfishly* in the sense that he attempts to choose a route of minimum delay. A Nash equilibrium in this setting is a solution wherein no agent has an incentive to deviate from his current strategy as he already travels from his origin to his destination along the shortest latency path possible. Roughgarden [101] analyzed the price of anarchy of routing games and revealed that it is independent of the network topology and only depends on the type of latency functions. Roughgarden and Tardos [102] showed that the price of anarchy is at most $4/3$ in case all latency function of the network are linear. The matching lower bound example is given by a simple instance consisting of two parallel arcs, known as the *Pigou instance* due to Pigou [97]. The example below illustrates that the Pigou instance achieves a price of anarchy of $4/3$.

Example 3.4.2. *Pigou instance (taken from [105]).* Consider the parallel-arc network in Figure 3.8. Agents want to traverse from the source s to the sink t. For both arcs $a \in \{1, 2\}$, there is a latency function $l_a(x_a) : \mathbb{R}_+ \to \mathbb{R}_+$, representing the travel time or latency for arc a depending on the flow x_a traversing the arc. The upper arc 1 has latency $l_1(x_1) = 1$, i. e., the latency is independently of the amount of flow on that arc. The lower arc has latency function $l_2(x_2) = x_2$, i. e., the latency grows linearly with the amount of flow on that arc. Suppose one wants to send one unit of flow from s to t and that this one unit of flow corresponds to infinitely

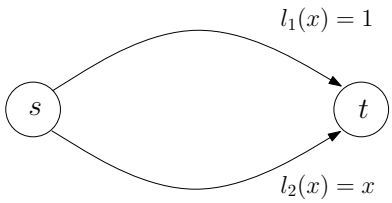

Figure 3.8: The Pigou instance, due to Pigou [97], where one unit of flow needs to be send from the source s to the sink t.

many agents that want to travel from s to t. Every (selfish) agent will reason as follows: The latency of the upper arc is one (independently of the flow) while the latency of the lower arc is at most one (and even strictly less than one if some agents are not using this arc). Thus, every agent chooses the lower arc. The resulting flow is a Nash equilibrium. Since every agent experiences a latency of one, the total latency of this Nash equilibrium is one. The optimal solution is given by $x_1 = x_2 = 1/2$ resulting in a total latency of $1 * (1/2) + (1/2)^2 = 3/4$. Thus, the price of anarchy of the Pigou instance equals $4/3$ matching the upper bound given by Roughgarden and Tardos [102] for the price of anarchy in case of linear latency functions. □

3.4.2 Smoothing the latency functions per arc

The Pigou instance given in Example 3.4.2 is the worst case lower bound and its price of anarchy matches the upper bound. The Pigou instance though seems fragile to perturbations in latency functions. For example, if there is one agent which does not exactly experience a latency of x_2 on arc 2 but a little more, say $x_2 + \epsilon$, then this agent will deviate to the first arc and by doing so improves his own latency as well as the latency of all other agents.

Buriol et al. apply the concept of smooth analysis to routing games, and in particular, to the Pigou instance (and some of its variants). Their motivation stems from the observation that in practice the traveling times along arcs are hardly exact as they are subject to (small) fluctuations. Such fluctuations might be caused by various reasons such as roadworks, accidents, weather conditions, varying driver behavior, and so on. The smoothing model they adopt perturbs the latency function $l_a(x)$ on each arc a by a factor $(1 + \epsilon_a)$, where ϵ_a is chosen randomly uniform at random out of the range $[0, \phi]$, for all arcs a. Note that now ϕ is the parameter determining how much noise is present in the smoothed instance (as opposed to $1/\phi$ in the previous section of this chapter). Let $\mathcal{I}$ denote a realization of the smoothed instance Φ. Further, let $\mathrm{NE}(\mathcal{I})$ denote the set of all Nash equilibria on the realization $\mathcal{I}$. The function $c(x)$ denotes the costs of a solution x, i. e.,

$$c(x) = x_1 \cdot (1 + \epsilon_1) \cdot l_1(x_1) + (1 - x_2) \cdot (1 + \epsilon_2) \cdot l_2(1 - x_2) = (1 + \epsilon_1) \cdot x_1 + (1 + \epsilon_2) \cdot (1 - x_2)^2.$$

Finally, denote by $x^{NE} = x^{\mathrm{NE}(\mathcal{I})}$ and x^* a Nash equilibrium and the optimal solution respectively. On the Pigou instance, Buriol et al. show the following upper bound on the smoothed price of anarchy

$$\mathop{\mathbf{E}}_{\mathcal{I}\sim\Phi}\left[\max_{x^{NE}\in\mathrm{NE}(\mathcal{I})} \frac{c(x^{NE})}{c(x^*)}\right] \leq \frac{12 + 4\phi}{12 + 6\phi + \log(1 + \phi) \cdot (1 - (1 + \phi)^3)/\phi^2}.$$

When ϕ tends to zero, the smoothed analysis converges to worst case analysis. Indeed, when ϕ tends to zero, the smoothed price of anarchy tends to the traditional price of anarchy, that is,

$$\lim_{\phi \to 0} \mathop{\mathbf{E}}_{\mathcal{I} \sim \Phi} \left[\max_{x^{NE} \in \mathrm{NE}(\mathcal{I})} \frac{c(x^{NE})}{c(x^*)} \right] = 4/3 \,.$$

Although it is to be expected that the smoothed price of anarchy is lower than the traditional price of anarchy, the authors conclude that their smoothing model does not yield a smoothed price of anarchy which significantly undercuts the traditional price of anarchy. Indeed, when setting $\phi = 1/2$, which allows for a relative large deviation of a $1/2$ with respect to the total flow which only equals 1, one finds that the smoothed price of anarchy still is relatively large:

$$\mathop{\mathbf{E}}_{\mathcal{I} \sim \Phi} \left[\max_{x^{NE} \in \mathrm{NE}(\mathcal{I})} \frac{c(x^{NE})}{c(x^*)} \right] \approx 1.256 < 4/3 \,.$$

3.4.3 Smoothing the latency functions per agent

I propose a different smoothing model where the latency function per agent is smoothed. In the preceding smoothing model designed by Buriol et al. the latency functions $l_a(x)$ are smoothed by a factor $(1 + \epsilon_a)$, for all arcs a. In such a setting though, each agents experiences each arc still in the same way. Moreover, for the worst case Pigou instance, only two factors are being smoothed, namely the latency functions of arcs 1 and 2. Combining these two observations, their smoothing model is, in my opinion, not that powerful. On any real life traffic network I would consider it unrealistic if every agent traversing an arc experiences exactly the same traveling time. I would say that each agents experiences slightly different circumstances, resulting in a slightly different latency perturbation.

I suggest a more powerful smoothing model which perturbs the latency each agent experiences on each edge. To this end, I split the one unit of flow which has to be routed in a large number of equal sized agents. That is, the one unit of flow is split in n flows, all of size $1/n$, where n is large. Let n_1 and n_2 be the number of agents which traverse arcs 1 and 2 respectively such that $n_1 + n_2 = n$. Then, the latency experienced by agent j on edges 1 and 2 respectively is given by:

$$\begin{aligned} l_{1j} &= (1 + \epsilon_{1j}) * 1 \\ l_{2j} &= (1 + \epsilon_{2j}) * (n_2/n), \end{aligned}$$

where ϵ_{1j} and ϵ_{2j} are independent random variables uniformly distributed over the range $[-\phi; \phi]$ for all agents $j \in \{1, \ldots, n\}$. Note that by definition it needs to be that $\phi \leq 1$. Further, if ϕ tends to zero, then the smoothing model tends to the traditional worst case analysis.

The result which I present below should be interpreted as a first indication that smoothing presents a powerful tool to study the quality of Nash equilibria in practice.

For simplicity of the subsequent analysis, let $\phi = 1/2$. Let $x_1 = n_1/n$ and $x_2 = n_2/n$ such that $x_1 + x_2 = 1$. First, I derive an upper bound on the expected total latency of a Nash equilibrium. Let x^{NE} be a Nash equilibrium and let N_a^{NE} be the corresponding set of agents who traverse edge a. Then, by definition of the Nash equilibrium, it is required that

$$(1 + \epsilon_{1j}) \leq (1 + \epsilon_{2j})\, x_2^{NE} \qquad \forall\, j \in N_1^{NE} \tag{3.8}$$

$$(1+\epsilon_{2j})\,x_2^{NE} \leq (1+\epsilon_{1j}) \qquad \forall\, j \in N_2^{NE} \tag{3.9}$$

To derive an upper bound on the expected total latency, I first determine the probability of x_2^{NE} exceeding some constant $\rho \in [0,1]$. First, note that $\mathbf{Pr}[x_2^{NE} \geq 1/3] = 1$ (as $\phi = 1/2$). By contradiction, assume that $x_2^{NE} < 1/3$. Then, there exist agents traversing the first arc and for each agent inequality (3.8) applies. Thus,

$$(1+\epsilon_{1j}) \leq (1+\epsilon_{2j})\,x_2^{NE} < (1+\epsilon_{2j})\,\frac{1}{3} \leq (1+\phi)\,\frac{1}{3} = \frac{1}{2},$$

which is a contradiction to $(1+\epsilon_{1j}) \geq 1-\phi = \frac{1}{2}$. In the remainder, assume that $x_2^{NE} \geq \frac{1}{3}$. When $x_2^{NE} \geq \rho \geq \frac{1}{3}$, then there exist at least ρn agents j for which

$$(1+\epsilon_{2j})\,\rho \leq (1+\epsilon_{1j}) \tag{3.10}$$

holds. Let N^ρ define the set of agents satisfying (3.10). Note in case $|N^\rho| > \rho n$, a Nash equilibrium having at least ρn agents traversing arc 2 can always be created by assigning all agents in N^ρ to arc 2 and letting the agents move from arc 2 to arc 1 if they can reduce their latency by doing so. Before continuing, I derive the probability that a random agent satisfies (3.10), given $\rho \geq \frac{1}{3}$.

$$\begin{aligned}
&\mathbf{Pr}[(1+\epsilon_{2j})\,\rho \leq (1+\epsilon_{1j})] \\
&= \mathbf{Pr}[(1+\epsilon_{2j})\,\rho \leq (1+\epsilon_{1j}) \,\Big|\, (1+\epsilon_{2j})\,\rho < \frac{1}{2}]\ \mathbf{Pr}[(1+\epsilon_{2j})\,\rho < \frac{1}{2}] \\
&\quad + \mathbf{Pr}[(1+\epsilon_{2j})\,\rho \leq (1+\epsilon_{1j}) \,\Big|\, (1+\epsilon_{2j})\,\rho \geq \frac{1}{2}]\ \mathbf{Pr}[(1+\epsilon_{2j})\,\rho \geq \frac{1}{2}] \\
&= 1 * \mathbf{Pr}[\epsilon_{2j} < \frac{1}{2\rho} - 1] + \int_{\frac{1}{2\rho}-1}^{\frac{1}{2}} \int_{\rho(1+\epsilon_{2j})-1}^{\frac{1}{2}} 1\,\partial\epsilon_{1j}\,\partial\epsilon_{2j} * \mathbf{Pr}[\epsilon_{2j} \geq \frac{1}{2\rho} - 1] \\
&= \frac{1}{2}\left(\frac{1}{\rho} - 1\right) + \frac{1}{2}\left(3 - \frac{1}{\rho}\right)\left(\frac{9}{4} - \frac{9\rho}{8} - \frac{5}{8\rho}\right).
\end{aligned}$$

For brevity, let $q(\rho) = \mathbf{Pr}[(1+\epsilon_{2j})\,\rho \leq (1+\epsilon_{1j})]$. Finally, I can derive the probability that $x_2^{NE} \geq \rho$:

$$\mathbf{Pr}[x_2^{NE} \geq \rho] = \mathbf{Pr}[|N^\rho| \geq \rho] = \sum_{k=\rho n}^{n} \binom{k}{n} [q(\rho)]^k\,[1-q(\rho)]^{n-k}$$

The probability $\mathbf{Pr}[x_2^{NE} \geq \rho]$ for different values is depicted in Figure 3.9. For $\phi = 1/2$, x_2^{NE} is no more than 0.75 with high probability. Since $\mathbf{E}\left[x_2^{NE}\right] = \int_0^1 \mathbf{Pr}[x_2^{NE} \geq \rho]\,\partial\rho$ it follows that $\mathbf{E}\left[x_2^{NE}\right] \approx 0.68$ yielding an expected total latency of 0.78 for the Nash equilibrium x^{NE}. The price of anarchy then amounts to only 1.04. Thus, if a agent's latency is subject to quite some noise, it may be expected that the quality of a Nash equilibrium is almost as good as the quality of the optimal solution (which is not necessarily a NE). I performed the above calculations for two more values of ϕ. The probability $\mathbf{Pr}[x_2^{NE} \geq \rho]$ for $\phi \in \left\{\frac{1}{4}, \frac{1}{10}\right\}$ has been depicted in Figure 3.9 as well. Naturally, adding less noise to the worst case instance implies a higher expected value of x_2^{NE}. For $\phi = \frac{1}{4}$ and $\phi = \frac{1}{10}$, $\mathbf{E}\left[x_2^{NE}\right]$ amounts to 0.73 and 0.85

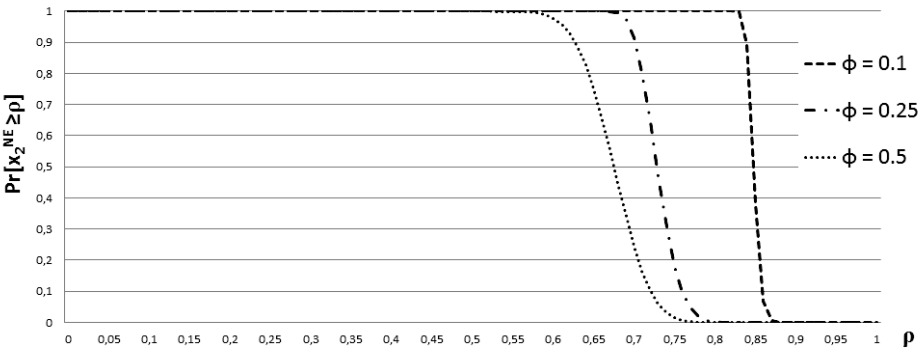

Figure 3.9: The probability $\mathbf{Pr}[x_2^{NE} \geq \rho]$ for three different values of the smoothing parameter ϕ.

yielding a price of anarchy of 1.07 and 1.17, respectively. When setting $\phi = 1/4$, I can compare myself to the result of Buriol et al. since the same amount of noise is added to the latency functions. The smoothing model applied in this work shows to be more powerful yielding a smoothed price of anarchy of only 1.07 versus their value of 1.26.

Although the result shown in this subsection is still premature, I hope it will foster future research to study Nash equilibrium not only in their worst case but also consider their average performance or their performance when some of the data has been smoothed.

3.5 Concluding remarks

This chapter shows that the lower bounds for all scheduling variants with restricted machines are rather robust against random noise, not only in expectation but even with high probability. The situation looks much better though for unrestricted machines where performance guarantees are in $\Theta(\phi)$ and $\Theta(\log \phi)$ for the jump and lexjump neighborhood, respectively. The latter bound also holds for the price of anarchy of routing on parallel links and for the list scheduling algorithm. Finally, I hinted that there is good hope that the smoothed price of anarchy for many games will undercut the traditional price of anarchy, matching better with the observed quality of Nash equilibrium in practice. Possible games could be routing games or weighted set cover.

There are several interesting directions for future research and I view the results presented in this chapter only as a first step towards fully understanding local search and greedy algorithms in the framework of smoothed analysis. For example, I have only perturbed the processing requirements, and it might be the case that the worst case bounds for the restricted scheduling variants break down if the sets $\mathcal{M}_j$ are subject to some degree of randomness as well. In general, it would be interesting to study different perturbation models where the sets $\mathcal{M}_j$ and/or the speeds s_i are perturbed. Lemma 3.2.16 indicates that there need to exist many machines having exponentially small speeds. I conjecture that if speeds are smoothed the smoothed performance guarantee lexjump optimal schedules on restricted related parallel machines is in $\Theta(\log \phi)$ as well.

Another direction to investigate is the following: since one does not know which local optimum is reached, I have always looked at the worst local optimum. I believe it would be interesting to study the 'average' lexjump optimal schedule. A final extension which I find worthwhile studying is the quality of coordination mechanisms under smoothing.

Chapter 4

Realtime scheduling on unrelated machines

The sporadic task model is a model of recurrent processes in hard real-time systems that has received great many attention in the last years; see for example [10, 13] and references therein. In this chapter, I study the problem of partitioning sporadic tasks to unrelated machines such that the tasks on each machine can be feasibly scheduled. Despite its importance for modern realtime systems, this problem has not been studied before. I present a polynomial time algorithm which approximates the problem with a constant speedup factor of $8 + 6\sqrt{2/3} \approx 12.89$ and show that any polynomial time algorithm needs a speedup factor of at least 2, unless $\mathcal{P} = \mathcal{NP}$. Further, a polynomial time approximation scheme is presented for the case when the number of machines is constant. Each of these algorithms applies the simple Earliest Deadline First policy as its run-time algorithm. Hence, once a partition has been determined, a simple and fast policy is applied to come up with a good schedule. This chapter is based on collaborative research with Alberto Marchetti-Spaccamela, Suzanne van der Ster and Andreas Wiese, [87], performed during my visit at La Sapienza, Università di Roma.

4.1 Introduction

Problem Definition. Let me briefly recapture the notion of sporadic task systems as they were introduced already in Subsection 1.2.3. A *sporadic task* $\tau = (c_\tau, d_\tau, p_\tau)$ is characterized by a worst-case execution time c_τ, a relative deadline d_τ, and a minimum interarrival separation, also known as the task's period, p_τ. Such a sporadic task generates a potentially infinite sequence of jobs with successive job arrivals separated by at least p_τ time units, it has an execution requirement less than or equal to c_τ and a deadline that occurs d_τ time units after its arrival time. A *sporadic task system* is comprised of several such sporadic tasks. Since the actual interarrival times can vary, there are infinitely many job sequences that can be generated.

A sporadic task system is said to be *feasible* upon a specified platform if it is possible to schedule the system on the platform such that all jobs of all tasks will meet their deadlines, under all permissible combinations of job arrival sequences by the different tasks comprising

the system.

An immediate application of such feasibility question in a hard realtime environment can be found in the airline industry. Consider for example the operating of an aircraft, an application which was also mentioned on the opening page of this dissertation. Several tasks need to be performed by the computer system of an aircraft to guarantee safe operation, for example for maintaining speed and altitude. These tasks need to be repeated over and over, but also need to be finished in a timely manner. Therefore, the challenge is to ensure that the processing power of the computer system is large enough to be able to feasibly schedule all tasks.

Determining whether a set of sporadic tasks can be feasibly scheduled is tough for various reasons. A single task spawns an infinite sequence of jobs which require scheduling over a long or even infinite time horizon. Therefore, it is not sufficient to only consider the first or even the first few jobs of a task, since two tasks might collide only after a long period of time. Secondly, a task which is small in execution time can still be tough to schedule in case its deadline is small as well. For example, consider a task τ which has small execution time c_τ and has its relative deadline equal to its execution time, i. e., $d_\tau = c_\tau$. When a job of this task is released, it needs to be scheduled immediately and to be processed without interruption. Therefore, this task blocks a processor for c_τ time units every time it is being released, and there is no freedom to move it around. Having two such tasks on a single machine will be infeasible, even if the machine were running at speed $2 - \epsilon$. Their aggregated utilization (defined by the processing time over the period; expressing the average use a task makes of the machine's capacity), however, may have been minor.

The above intuition is confirmed by the fact that, even on a single machine, the problem of scheduling a sporadic task system is known to be *co-$\mathcal{NP}$-hard* [43]. On the positive side, it is known that the Earliest Deadline First (EDF) algorithm (introduced in Subsection 1.3.1) which schedules at any time the job with the earliest absolute deadline is *optimal* on a single processor [83]. Here, optimality means that for any feasible job sequence, EDF produces a feasible schedule. Although one can validate whether a task system is feasible by running EDF, this does not provide an efficient feasibility test for scheduling a task system to a single machine. The reason is that in the worst case the infeasibility only occurs at a late point in the schedule, say close to the hyperperiod (which is the least common multiple of the tasks' periods). Then, the scheduler would have to evaluate the deadline of each job which was released up to that point in time. Consequently, executing EDF might run in exponential time. Moreover, no efficient condition has been derived that returns whether the schedule produced by EDF will be feasible or not, in time shorter than the running time of EDF.

On multiprocessor systems, there are two main paradigms for scheduling: *global* and *partitioned* scheduling. In the former, all tasks can use all machines, and jobs can even be migrated from one machine to another. In the partitioned scheduling approach each task has to be assigned to one of the machines such that all its jobs have to be executed on this specific machine. Since the process of partitioning tasks among processors reduces a multiprocessor scheduling problem to a series of single processor problems. In the current chapter (as well as the next) I study partitioned scheduling, as in practice this setting is mostly used. Partitioned scheduling is easier to implement and does not require the communication overhead which is needed if a single task is split between multiple processors. Such communication overhead not only slows down the system but might additionally lead to security issues, for instance in

the airline industry.

As EDF is an optimal policy for preemptive single processor scheduling, the algorithm is often used as the run-time scheduling algorithm on each machine. Further, EDF has been shown to be a simple but powerful and hence often applied algorithm in practice. Therefore, in the upcoming two chapters, EDF will also be my run-time scheduling algorithm of choice. I remark here that though I use EDF as the run-time scheduling algorithm, EDF is not part of the proposed feasibility test since that would yield an exponential time test. Instead, the test makes use of conditions that should be satisfied by any EDF schedule and which can be checked in appropriate time.

In recent years, hardware design has seen a highly visible trend towards heterogeneous processors. In particular, modern hardware architectures often contain specialized processors for certain tasks (e. g., graphical processors, floating-point units). To model the actual behavior of the different types of processors when making scheduling decisions, this chapter assumes that the given machines are *unrelated*; i. e., it is assumed that the processing time of each task depends on the machine where it is executed, including the possibility that some tasks cannot be executed on some machine at all.

Since it is hard to determine whether a task system can be feasibly be scheduled to even a single machine, the focus is on designing approximate feasibility tests that run efficiently but introduce an error in the decision process, controlled by an accuracy parameter α (the speedup). If a single machine *α-approximation test* returns *feasible*, then the task system is guaranteed to be feasible on an α-speed processor(s); if the test returns *infeasible*, the task system is guaranteed to be infeasible on a unit-speed processor(s).

Related Work. For the case of scheduling sporadic tasks to a single processor, a FPTAS feasibility test for EDF has been proposed [28] (i.e. for any $\epsilon > 0$, there exists a $(1+\epsilon)$-approximation test with running time polynomial in the number of tasks and in $1/\epsilon$). For an overview of more results on the feasibility analysis of sporadic task systems on a single processor I refer the reader to Baker and Baruah [10].

For realtime scheduling on multiple machines, as far as I am aware of, only the case of identical parallel machines has been studied. Most results deal with the global scheduling paradigm, see e. g., [14, 19, 61]. It is known that the natural EDF-policy is no longer optimal, but that any feasible collection of tasks on m machines of unit speed is schedulable using EDF on m machines of speed $2 - \frac{1}{m}$ [95]. Also, a corresponding test for task sets is known [13, 21]. Recently, Anand et al. [3] presented an online algorithm needing only a speedup factor of $e/(e-1) \approx 1.58$.

For the partitioned scheduling paradigm, most of prior research has considered the special case of implicit deadline systems in which all tasks have their deadlines equal to their period parameters (i. e., $d_\tau = p_\tau$ for all τ). In this case, a set of tasks on a machine is feasible if and only if the sum of the utilizations c_τ/p_τ are at most one and the problem reduces to BIN PACKING. Recall that for bin packing an asymptotic FPTAS exists [73]. For implicit deadline systems, several sufficient feasibility tests have been derived for EDF and various alternative scheduling policies, see [10]. In case that deadlines are not constrained, much less is known. Baruah and Fisher [11] propose an algorithm which can partition any set of tasks that is feasible on m machines such that the assignment is feasible if the machines run $(4 - \frac{2}{m})$ times

faster. In [48] a similar result is given if the tasks are scheduled according to static priorities, rather than with the more powerful EDF-policy. Chen and Chakraborty [30] improved upon these results by showing that a deadline-monotonic policy with approximate demand bound functions leads to a $\frac{3e-1}{e} - \frac{1}{m} \approx (2.6322 - \frac{1}{m})$-approximation test in case of constrained deadlines ($d_\tau \leq p_\tau$ for all $\tau \in \mathcal{T}$) and a $(3 - \frac{1}{m})$-approximation test for unconstrained deadlines. When taking the number of needed machines as objective function, in [42] a PTAS has been proposed for the case where tasks are scheduled according to fixed priorities using resource augmentation. Also, the existence of an asymptotic FPTAS has been ruled out, thus showing that the problem is indeed harder than BIN PACKING.

As mentioned above, I distinct from previous work by assuming machines to be unrelated. On unrelated machines the only results known are for the traditional machine scheduling problem with makespan minimization. Lenstra, Shmoys and Tardos [78] gave a polynomial time 2-approximation algorithm for the problem of minimizing the makespan of a set of jobs and that it is $\mathcal{NP}$*-hard* to achieve a performance ratio strictly less than 1.5. Despite a lot of effort, the only improvements in the setting of an arbitrary number of machines are a 1.75-approximation algorithm for the graph balancing case [41] and a $33/17 \approx 1.94$-approximation algorithm for the restricted assignment case [114]. Azar and Epstein [8] consider ℓ_p norms (for finite $p > 1$, rather than the makespan) for which they improved the previously best result of $\theta(p)$ [6] to a 2-approximation and even a $\sqrt{2}$-approximation for the ℓ_2 norm. This was improved to a better-than-two result for all $p > 1$ in [76]. For a constant number of machines polynomial time approximation schemes are known [72, 78]; recently a PTAS has been proposed for the case that each machine belongs to one of a fixed number of types, and processing time of each job depends only on the job and the type of the machine it is assigned to [22].

Contribution. To the best of my and my co-authors' knowledge, no non-trivial algorithm is known for assigning a set of sporadic tasks to a set of unrelated machines. First, I present an algorithm that, given a task system for which a task assignment on m machines exists, finds a task assignment that can be scheduled on m machines that are $11 + 4\sqrt{3} \approx 17.9$ times as fast. Afterwards, I improve upon this result by showing a second algorithm, which is built upon partially different techniques, that 'only' needs a speedup factor of $8 + 6\sqrt{2/3} \approx 12.89$. Also, I show that no polynomial time algorithm can compute a task assignment needing a speedup factor of $2 - \epsilon$ for $\epsilon > 0$, unless $\mathcal{P} = \mathcal{NP}$. Note that this bound is stronger than the best known $(3/2 - \epsilon)$-hardness result for the contained problem of minimizing the makespan when scheduling jobs on unrelated machines [78].

In case that the number of machines is fixed, a polynomial time algorithm is presented which either finds a feasible task assignment on m machines that are $(1 + \epsilon)$ times as fast, or guarantees that no solution exists on unit-speed processors.

In order to be able to achieve these results, I provide a deep understanding of the *demand bound function* (dbf) which yields a necessary and sufficient condition for a task system to be feasible on one machine. In particular, I present two new relaxations for handling this well-studied function. To obtain the result on an arbitrary number of machines, I give a set of sparse linear constraints which approximate the dbf up to a constant factor. Due to the sparsity and by using other techniques I design an efficient iterative rounding procedure. For the case of a constant number of machines I observe that - since a task might release

an infinite sequence of jobs - one cannot exploit the technique of partitioning the task set into 'big' and 'small' tasks as is done in traditional job scheduling problems. In particular, one cannot simply enumerate those. An important feature of the proposed dbf-relaxation is that the feasibility test of assigning a task with deadline D to a machine having tasks with deadlines less than D already assigned to it, requires only a constant number of quantities of the previously assigned tasks. Exploiting this feature along with applying some other tricks enables me to polynomially bound the running time of a dynamic programming algorithm.

Outline. In Section 4.2 I formally introduce the demand bound function, the problem setting and some needed theory on integer linear programming. In Sections 4.3 and 4.4 the results for an arbitrary number of machines and a constant number of machines are presented respectively.

4.2 Preliminaries

Given is a set M of m unrelated parallel machines and a sporadic task system T, with $|T| = n$. As I consider unrelated machines, the processing requirement of a task is machine dependent and is denoted by $c_{i,\tau}$ instead of just c_τ as in standard realtime scheduling. I assume all parameters to be integer and strictly positive. In particular, this assumption implies that $d_\tau \geq 1$ for all tasks τ. I study the problem of finding a partition $\mathcal{T} = \{\mathcal{T}_i\}_{i \in M}$ of the tasks over the machines such that $\cup_i \mathcal{T}_i = T$ and $\mathcal{T}_i \cap \mathcal{T}_{i'} = \emptyset$ for any two machines $i \neq i'$. Note that jobs may be preempted and resumed at a later point in time (but on the same machine). The central question in this chapter is whether a partition exists which yields a feasible schedule in case EDF is applied as a run-time algorithm on each machine. EDF is the algorithm of my choice since, once a partition is known, the scheduling problem reduces to collection of single machine scheduling problems for which EDF is known to be optimal. From now on, I also speak of a task assignment $\mathcal{T}$ instead of a partition.

Definition 4.2.1. *A task assignment is* feasible *if any job arrival sequence of the tasks in $\mathcal{T}_i$ can be feasibly scheduled on i according to the Earliest Deadline First algorithm, for all machines $i \in M$.*

Definition 4.2.2. *An* α-approximation test *for the problem of assigning tasks to unrelated machines is an algorithm that runs in polynomial time and which either (i) guarantees that there is no feasible partition of the tasks to the given machines (running at unit speed), or (ii) finds an assignment which is feasible if the machines run at speed α.*

I denote by $u_{i,\tau} = c_{i,\tau}/p_\tau$ the *utilization* of task τ on machine i. Given a task assignment $\mathcal{T}$, the utilization of a machine i is given by $u_i = u_{i,\mathcal{T}} = \sum_{\tau \in \mathcal{T}_i} u_{i,\tau}$.

Property 4.2.3. *In a feasible assignment $\mathcal{T}$, $u_i \leq 1$ for all $i \in M$ is a necessary but not sufficient condition for feasibility, see [83].*

Clearly, the necessary condition $u_i \leq 1$ for all $i \in M$ implies that if there is a task τ such that $u_{i,\tau} > 1$, this task will never be assigned to machine i.

Property 4.2.4. *Task τ will in a feasible task assignment $\mathcal{T}$ not be assigned to machine i if $u_{i,\tau} > 1$.*

Further, if for task τ and machine i it holds that $c_{i,\tau} > d_\tau$, then τ will neither be assigned to machine i as already the first job of τ will miss its first deadline at d_τ.

Property 4.2.5. *Task τ will in a feasible task assignment $\mathcal{T}$ not be assigned to machine i if $c_{i,\tau} > d_\tau$.*

The *synchronous arrival sequence* for task system T is defined to be the collection of job arrivals in which each task in T generates a job at time-instant zero, and subsequent jobs arrive as soon as legally permitted, i. e., task τ generates a job at each time-instant $k \cdot p_\tau$ for $k = 0, 1, 2, \ldots$.

It follows from Baruah, Mok and Rosier [15] that the following condition is both necessary and sufficient for the feasibility of a task system on a single machine:

Property 4.2.6. *A set of sporadic tasks $\mathcal{T}_i$ is EDF-schedulable on machine i if and only if the following two conditions are both satisfied:*

1. *the utilization of the task system does not exceed 1, i. e.,*

$$u_i = \sum_{\tau \in \mathcal{T}_i} u_{i,\tau} \leq 1, \text{ and}$$

2. *all jobs with deadlines in $[0, lcm_{\tau \in \mathcal{T}_i} \{p_\tau\}]$ in the synchronous arrival sequence of $\mathcal{T}_i$ meet their deadlines, where lcm denotes the least common multiple.*

The first condition can be checked in linear time. Checking the second condition implies that in worst case the deadline of each job released before $lcm_{\tau \in \mathcal{T}_i} \{p_\tau\}$ should be checked. If none of the tasks' periods divide each other, then the value $lcm_{\tau \in \mathcal{T}_i} \{p_\tau\}$ might be exponentially large with respect to the smallest periods. Therefore, the smallest tasks might release exponentially many jobs before time $lcm_{\tau \in \mathcal{T}_i} \{p_\tau\}$ which gives also exponentially many deadlines to check. Therefore, the above feasibility condition yields an exponential time test to check whether $\mathcal{T}_i$ is EDF-schedulable. The following necessary and sufficient condition for a task system T to be feasible can be derived from Property 4.2.6. Define the demand bound function $\mathrm{dbf}_{i,\tau}(s)$ of a task τ on a machine i at time s as

$$\mathrm{dbf}_{i,\tau}(s) := \max \{0;\ \lfloor (s + p_\tau - d_\tau)/p_\tau \rfloor\, c_{i,\tau}\}\,.$$

The function $\mathrm{dbf}_{i,\tau}(s)$ counts the number of jobs of task τ which have to processed by machine i at time s. If less than $\mathrm{dbf}_{i,\tau}(s)$ jobs are completed by time s, then one of these jobs has definitely missed its deadline making the schedule infeasible. It follows from Baruah et al. [15] that for unrelated parallel machines:

Property 4.2.7. *An assignment $\mathcal{T} = \{\mathcal{T}_i\}_{i \in M}$ is feasible for task system T if and only if for all $i \in M$, and for all $s > 0$*

$$dbf_{i,\mathcal{T}}(s) := \sum_{\tau \in \mathcal{T}_i : d_\tau \leq s} dbf_{i,\tau}(s) = \sum_{\tau \in \mathcal{T}_i : d_\tau \leq s} \max \left\{ 0;\ \left\lfloor \frac{s + p_\tau - d_\tau}{p_\tau} \right\rfloor c_{i,\tau} \right\} \leq s.$$

I will write dbf_i instead of $\text{dbf}_{i,\mathcal{T}}$ whenever the assignment $\mathcal{T}$ is clear from the context.

The final preliminary remarks are on some insights gained from linear programming theory. Lemma 4.2.8 implies that in polynomial time a solution to a linear program (LP) can be derived for which the number of strictly positive variables is no more than the number of constraints in the program. I will intuitively explain the intuition behind the lemma.

Consider a polyhedron $R = \{\mathbf{x} \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$ where $\mathbf{x}$ is a vector of dimension k and $\mathbf{b}$ is a vector of dimension l. It follows that R is the intersection of $k + l$ halfspaces. An extreme point $\mathbf{x}^*$ of R is a point in R such that either $\mathbf{x}^* + \mathbf{y}$ or $\mathbf{x}^* - \mathbf{y}$ is no longer in R, for any vector $\mathbf{y}$ of dimension k. Further, a set of vectors $\mathbf{a}_1$ up to $\mathbf{a}_l$ of the same dimension is called linearly independent if there do not exist constants α_1 up to α_l such that

$$\alpha_1 \cdot \mathbf{a}_1 + \ldots + \alpha_l \cdot \mathbf{a}_l = \mathbf{0}.$$

Finally, call $\mathit{rank}(\mathbf{A})$ the maximum number of linearly independent rows of the matrix $\mathbf{A}$. It is easy that show that $\mathit{rank}(\mathbf{A})$ then also equals the maximum number of linearly independent columns of the matrix $\mathbf{A}$. From linear algebra theory it follows that an extreme point $\mathbf{x}^*$ of R satisfies $\mathit{rank}(\mathbf{A})$ of the $k + l$ inequalities defining R with equality. Therefore, any extreme point $\mathbf{x}^*$ of R can have at most $\mathit{rank}(\mathbf{A})$ strictly positive variables.

Lemma 4.2.8 (Rank Lemma)**.** *Given a polyhedron $R = \{\mathbf{x} \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$ and let $\mathbf{x}^*$ be an extreme point of R such that $x_j^* > 0$ for all j. Then, the number of variables is equal to the number of linearly independent rows of the matrix $\mathbf{A}$, that is, $rank(\mathbf{A})$.*

Corollary 4.2.9 follows from the Rank Lemma and is used in the proofs of the upcoming Lemmas 4.3.2 and 4.3.5. The corollary follows as the rank $\mathit{rank}(\mathbf{A})$ of matrix $\mathbf{A}$ is upper bounded by the number of rows of the matrix $rows(\mathbf{A})$. The claim on the running time follows since for instance the ellipsoid method solves linear programming problems in polynomial time and as an optimal solution by definition needs to be an extreme point solution.

Corollary 4.2.9. *Consider a linear program $\min\{\mathbf{c}^T\mathbf{x} \mid \mathbf{x} \in R\}$ where the polyhedron R is given by $\{\mathbf{x} \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$. Then, one can derive in polynomial time an optimal solution $\mathbf{x}$ for which at most $rows(\mathbf{A})$ variables x_j are strictly positive. Such a solution $\mathbf{x}$ will be an extreme point of R.*

4.3 Arbitrary number of machines

In this section, I first present an $\alpha = 11 + 4\sqrt{3} \approx 17.9$-approximation test, and later a $\alpha = 8 + 6\sqrt{2/3} \approx 12.89$-approximation test, for assigning tasks to unrelated machines. The improved algorithm relies partially on different techniques and hence both algorithms are presented here. Further, I show that the problem is *$\mathcal{NP}$-hard* to approximate the scheduling problem with a ratio of $2 - \epsilon$, for any $\epsilon > 0$.

4.3.1 A constant factor approximation algorithm

The feasibility problem of assigning tasks to unrelated machines is formulated as a linear program such that the tasks on each machine can be feasibly scheduled using the EDF-scheduler. First, I derive a set of linear inequalities which are

- necessary, meaning that they are fulfilled by any feasible assignment,
- approximately sufficient, meaning that any assignment which (approximately) fulfills the constraints is feasible if the speeds of the machines are increased by some constant factor, and
- sparse, as each variable appears in only two inequalities.

I introduce a binary variable $y_{i,\tau}$ for each machine-and-task-pair to model whether task τ is assigned to machine i. Conform Properties 4.2.4 and 4.2.5, a variable $y_{i,\tau}$ will only be created if task τ can potentially be assigned to machine i, that is, if both $u_{i,\tau} \leq 1$ and $c_{i,\tau} \leq d_\tau$ hold true. The constraints introduced will capture the conditions stated by Property 4.2.6. The first set of constraints are utilization bounds on all tasks assigned to the same machine i. Formally, I demand that

$$\sum_{\tau \in T} u_{i,\tau} y_{i,\tau} \leq 1 \quad \forall \quad i \in M. \tag{4.1}$$

Secondly, I require for each machine that the sum of the processing requirements of the tasks assigned to this machine and whose deadlines are in the interval $(2^{k-1}, 2^k]$, is at most 2^k. Formally, I require

$$\sum_{\tau \in T : d_\tau \in (2^{k-1}, 2^k]} c_{i,\tau} y_{i,\tau} \leq 2^k \quad \forall \quad i \in M,\, k \in \mathbb{N}. \tag{4.2}$$

I will refer to these two conditions as the *relaxed dbf-constraints*. It is clear that these constraints have to be fulfilled by any feasible task assignment. Since they are linear, they can be used in an LP-relaxation for the problem. Their sparsity gives the potential to derive efficient rounding schemes which result in integral solutions, violating the relaxed dbf-constraints only by constant factors. In this section, I will present an algorithm that uses this type of rounding schemes. To this end, the following lemma shows that, even when violated up to a constant factor, the relaxed dbf-constraints are approximately sufficient.

Lemma 4.3.1. *Let $\mathcal{T}$ be an assignment for the task system T such that, for all machines i and all integers k,*

- $\sum_{\tau \in \mathcal{T}_i} u_{i,\tau} \leq \beta$*,*
- $\sum_{\tau \in \mathcal{T}_i : d_\tau \in (2^{k-1}, 2^k]} c_{i,\tau} \leq \beta \cdot 2^k$*,*
- *and Properties 4.2.4 and 4.2.5*

hold. Then $\mathrm{dbf}_{i,\mathcal{T}}(s) \leq 6\beta s$ for all $s \geq 0$ and $\mathcal{T}$ is a feasible assignment under a speedup factor of 6β.

Proof. Let $k \in \mathbb{N}$ and $s := 2^k$. Consider an assignment $\mathcal{T}$ of the tasks in T to the machines in M. For any machine $i \in M$, bound $\mathrm{dbf}_{i,\mathcal{T}}(s)$ by

$$\begin{aligned}\mathrm{dbf}_{i,\mathcal{T}}(s) &= \sum_{\tau \in \mathcal{T}_i : d_\tau \leq s} \left\lfloor \frac{s + p_\tau - d_\tau}{p_\tau} \right\rfloor c_{i,\tau} \leq \sum_{\tau \in \mathcal{T}_i : d_\tau \leq s} \left(s \frac{c_{i,\tau}}{p_\tau} + c_{i,\tau} \right) \\ &\leq s \sum_{\tau \in \mathcal{T}_i} \frac{c_{i,\tau}}{p_\tau} + \sum_{\tau \in \mathcal{T}_i : d_\tau \leq s} c_{i,\tau} = s \sum_{\tau \in \mathcal{T}_i} u_{i,\tau} + \sum_{\ell=0}^{\log_2(s)} \sum_{\tau \in \mathcal{T}_i : d_\tau \in (2^{\ell-1}, 2^\ell]} c_{i,\tau}\end{aligned}$$

$$\leq \beta s + \sum_{\ell=0}^{k} \beta \cdot 2^{\ell} \leq \beta s + \beta \cdot 2^{k+1} \leq \beta \cdot 3s.$$

The third last inequality follows from Property 4.2.4 and $c_{i,\tau} \leq d_\tau \leq 2^\ell$ by Property 4.2.5. Hence, for any machine i and for arbitrary s it is that

$$\mathrm{dbf}_{i,\mathcal{T}}(s) \leq \mathrm{dbf}_{i,\mathcal{T}}(2^{\lceil \log_2 s \rceil}) \leq 3\beta \cdot 2^{\lceil \log_2 s \rceil} \leq 6\beta s.$$

This implies that $\mathcal{T}$ is a feasible assignment for scheduling the tasks in T to the set of unrelated machines whenever the machines receive a speedup factor of 6β. □

Let $\rho > 1$ be some constant. Consider an instance of the scheduling problem. Define the function $r(x) := \rho^{\lceil \log_\rho x \rceil}$. Let $d_{\max} := \max_{\tau \in T} d_\tau$ and define the set $\mathcal{D}_\rho := \{\rho^0, \rho^1, \ldots, r(d_{\max})\}$. I formulate the following linear program, denoted by ASS-LP:

$$\sum_{i \in M} y_{i,\tau} = 1 \qquad \forall \tau \in T \tag{4.3a}$$

$$\sum_{\tau \in T} u_{i,\tau} y_{i,\tau} \leq 1 \qquad \forall i \in M \tag{4.3b}$$

$$\sum_{\tau \in T:\, r(d_\tau) = D} c_{i,\tau} y_{i,\tau} \leq D \qquad \forall D \in \mathcal{D}_\rho,\, \forall i \in M \tag{4.3c}$$

$$y_{i,\tau} \geq 0 \qquad \forall \tau \in T, \forall i \in M : u_{i,\tau} \leq 1 \wedge c_{i,\tau} \leq d_\tau \tag{4.3d}$$

If ASS-LP is infeasible, then there can be no feasible integral solution, that is, there can be no feasible partition of the tasks to the machines. Therefore, assume that ASS-LP is feasible and that a feasible solution is given by $\mathbf{y}^*$. For each machine i and deadline $D \in \mathcal{D}_\rho$ extract a value $U_{i,D} := \sum_{\tau \in T:\, r(d_\tau) = D} c_{i,\tau} y^*_{i,\tau}$. Based on these values, I define a variation of ASS-LP, denoted by ASS-LP2 in the sequel. ASS-LP2 follows from ASS-LP by replacing the constraints (4.3c):

$$\sum_{i \in M} y_{i,\tau} = 1 \qquad \forall \tau \in T \tag{4.4a}$$

$$\sum_{\tau \in T} u_{i,\tau} y_{i,\tau} \leq 1 \qquad \forall i \in M \tag{4.4b}$$

$$\sum_{\tau \in T:\, r(d_\tau) = D} c_{i,\tau} y_{i,\tau} \leq U_{i,D} \qquad \forall D \in \mathcal{D}_\rho,\, \forall i \in M \tag{4.4c}$$

$$y_{i,\tau} \geq 0 \qquad \forall \tau \in T, \forall i \in M : u_{i,\tau} \leq 1 \wedge c_{i,\tau} \leq d_\tau \tag{4.4d}$$

Clearly if $\mathbf{y}^*$ is a feasible solution for ASS-LP it is also a feasible solution for ASS-LP2, and if ASS-LP2 is infeasible then ASS-LP is infeasible as well implying that no feasible partition exists. I proceed by rounding $\mathbf{y}^*$ to an integral vector which *approximately* satisfies ASS-LP2. Namely, I follow an iterative rounding approach, similar to [71], which derives an integer solution $\hat{\mathbf{y}}$ that satisfies constraints (4.4a) and (4.4d) and the following two inequalities

$$\sum_{\tau \in T} u_{i,\tau} \hat{y}_{i,\tau} \leq 4 \qquad \forall i \in M \tag{4.5}$$

$$\sum_{\tau \in T:\, r(d_\tau) = D} c_{i,\tau} \hat{y}_{i,\tau} \leq U_{i,D} + 3D \qquad \forall D \in \mathcal{D}_\rho,\, \forall i \in M \tag{4.6}$$

The idea of the iterative rounding procedure is the following. In each iteration k, first compute an extreme point solution $\mathbf{y}^k$ of a linear program LP^k where LP^0 equals ASS-LP2 and each LP^k is obtained by fixing some variables and removing some constraints of LP^{k-1}.

Given LP^k and a corresponding fractional solution $\mathbf{y}^k$, the linear program LP^{k+1} is derived as follows. First fix all variables which are integral in $\mathbf{y}^k$, i. e., those variables are not allowed to be changed anymore in the remainder of the procedure. Then, check whether there exists a constraint of either type (4.4b) or type (4.4c), with at most three fractional variables. LP^{k+1} is then generated by dropping this constraint. If such a constraint does not exist, then an integral solution is obtained by rounding all variables of $\mathbf{y}^k$ in a suitable way. The key lemma in the procedure is given below. Recall that an extreme point solution to a linear program can be derived in polynomial time, see Corollary 4.2.9.

Lemma 4.3.2. *Let $\mathbf{y}^k$ be an extreme point solution to LP^k. Then either,*
(i) there is a machine i for which there is a constraint of type (4.4b) *in LP^k such that there are at most three tasks τ with $y_{i,\tau} \in (0,1)$ (i. e., $y_{i,\tau}$ is fractional), or*
(ii) there is a machine i and a deadline $D \in \mathcal{D}_\rho$ for which there is a constraint of type (4.4c) *in LP^k and there are at most three tasks τ with $r(d_\tau) = D$ and $y_{i,\tau} \in (0,1)$, or*
(iii) for each machine i there are exactly four tasks τ with $y_{i,\tau} \in (0,1)$.

Proof. Let $\mathbf{y}^k := \left\{y^k_{i,\tau}\right\}_{i\in M,\tau\in T}$ represent an extreme point solution of the solution space implied by the inequality system in LP^k. Let $I(\mathbf{y}^k)$ be the number of variables $y^k_{i,\tau}$ for which $y^k_{i,\tau} = 1$. Similarly, let $F(\mathbf{y}^k)$ be the number of variables $y^k_{i,\tau}$ for which $y^k_{i,\tau} \in (0,1)$. Further, let z_a, z_b and z_c respectively be the number of inequalities of type (4.4a), (4.4b) and (4.4c) in phase k of the iterative rounding procedure.

Each task $\tau \in T$ which is fractionally assigned, generates at least two variables $y^k_{i,\tau}$ which are fractional. Hence,

$$z_a = n = |T| \leq I(\mathbf{y}^k) + F(\mathbf{y}^k)/2. \tag{4.7}$$

The number of inequalities of type (4.4a), (4.4b) and (4.4c) is given by $z_a + z_b + z_c$. As $\mathbf{y}^k$ presents an extreme point solution, Corollary 4.2.9 yields that the number of nonzero variables $y^k_{i,\tau}$ is upper bounded by $z_a + z_b + z_c$, that is,

$$\#\text{ nonzeros } = I(\mathbf{y}^k) + F(\mathbf{y}^k) \leq z_a + z_b + z_c. \tag{4.8}$$

It follows that

$$F(\mathbf{y}^k) \overset{(4.8)}{\leq} z_a + z_b + z_c - I(\mathbf{y}^k) \overset{(4.7)}{\leq} z_b + z_c + F(\mathbf{y}^k)/2,$$

which yields $F(\mathbf{y}^k) \leq 2(z_b + z_c)$.

In case $z_c > z_b$, it follows that there are strictly less than $4z_c$ fractional variables $y^k_{i,\tau}$. Also, each variable $y^k_{i,\tau}$ appears in exactly one inequality of type (4.4c). Therefore, the average number of fractional variables $y^k_{i,\tau}$ per inequality of type (4.4c) is $(2z_b+2z_c)/z_c = 2(1+z_b/z_c) < 4$. Hence, there exists at least one inequality of type (4.4c), characterized by some machine $i \in M$ and some $D \in \mathcal{D}$, such that in this inequality there are at most three variables $y^k_{i,\tau}$ which are fractional, showing Case (ii).

Secondly, consider the second case $z_b > z_c$. Similar reasoning shows that there exists at least one inequality of type (4.4b), characterized by some machine $i \in M$, such that in this inequality there are at most three variables $y^k_{i,\tau}$ which are fractional, showing Case (i).

Finally, consider the situation where $z_c = z_b$ such that the number of fractional variables is at most $2(z_b + z_c) = 4z_b = 4z_c$. Since no constraint of type (4.4b) carries 3 fractional variables or less, it follows that each such constraint carries exactly $4z_b/z_b = 4$ fractional variables, yielding Case (iii). □

If Case (i) of Lemma 4.3.2 applies for a machine i , then LP^{k+1} is obtained by dropping the corresponding constraint

$$\sum_{\tau \in T} u_{i,\tau} y_{i,\tau} \leq 1.$$

Since integer variables have been fixed before, and as the variable $y_{i,\tau}$ only exists in case $u_{i,\tau} \leq 1$ (conform Property 4.2.4) it holds for any solution of the subsequent linear program that

$$\sum_{\tau \in T} u_{i,\tau} y_{i,\tau} \leq 4.$$

Therefore, although disregarding this constraint of type (4.4b) in the sequel, I preserve that the right-hand side of this constraint is violated by at most an amount of 3, no matter how the involved variables are rounded in the subsequent iterations.

Similarly, if Case (ii) applies for a machine i and a deadline D, then LP^{k+1} is obtained by dropping this constraint. Independent of how the involved variables are rounded in the subsequent iterations of the procedure, the dropped constraint will always satisfy

$$\sum_{\tau \in T:\, r(d_\tau) = D} c_{i,\tau} y_{i,\tau} \leq U_{i,D} + 3D,$$

as the variable $y_{i,\tau}$ only exists in case $u_{i,\tau} \leq 1$ and $c_{i,\tau} \leq d_\tau$ (conform Properties 4.2.4 and 4.2.5), and integer variables have been fixed before. Therefore, although disregarding this constraint of type (4.4c) in the sequel, I preserve that the right-hand side of this constraint is violated by at most an amount of $3D$.

Finally, if Case (iii) applies, all remaining tasks will be assigned at once using the lemma below.

Lemma 4.3.3. *Assume that in* $\mathbf{y}^k$ *for each machine* i *there are exactly four tasks* τ *such that* $y^k_{i,\tau} \in (0,1)$. *Then, a partition can be computed in polynomial time which assigns these tasks to the machines such that each machine is assigned at most two of them.*

Proof. Let T' denote the set of yet unassigned tasks. Construct a bipartite graph $G = (V \cup W, E)$ by introducing one vertex $v_\tau \in V$ for each task $\tau \in T'$ and two vertices $w_i^{(1)}, w_i^{(2)} \in W$ for each machine i. For each combination of a task τ and a machine i such that $y_{i,\tau} \in (0,1)$, I introduce either an edge $\{v_\tau, w_i^{(1)}\} \in E$ or an edge $\{v_\tau, w_i^{(2)}\} \in E$. In both cases, the respective edge receives a weight of $1/2$. This assignment can be done such that each vertex $w_i^{(\ell)}$ with $\ell \in \{1,2\}$ has at most degree two. By the assumption on the vector $\mathbf{y}^k$, this yields a fractional matching in which each vertex v_τ is completely assigned (or maybe even overassigned) and each vertex $w_i^{(l)}$ has at most a total weight of one assigned to it. By the

use of augmenting paths, one can derive in polynomial time an integral matching adhering this same property. The way to proceed here is to transform the fractional matching into any integral matching. In case there is a machine vertex $w_i^{(l)}$ having two tasks fully assigned to it, then feasibility requires that there is an augmenting path starting in this vertex $w_i^{(l)}$ and ending in another machine vertex to which no task has been assigned. Augmenting this path will decrease the number of machines to which two tasks are fully assigned by one. An important observation for the procedure to work is that any path leaving a machine vertex to which two task are assigned can neither arrive back at a vertex already visited by the path nor visit another machine to which two tasks are assigned. This observation follows as each vertex in the constructed bipartite graph is adjacent to exactly two edges only. To conclude, the final integral matching which is derived presents an assignment of the tasks T' to the machines such that each machine has at most two tasks from T' assigned to it. □

If either all constraints have been removed or if Lemma 4.3.3 has been applied, a task assignment $\hat{\mathrm{y}}$ is obtained which satisfies

$$\sum_{\tau \in T} u_{i,\tau} \hat{y}_{i,\tau} \leq 4 \quad \forall i \in M$$

$$\text{and} \sum_{\tau \in T:\, r(d_\tau)=D} c_{i,\tau} \hat{y}_{i,\tau} \leq U_{i,D} + 3D \quad \forall i \in M,\, D \in \mathcal{D}_\rho.$$

Observe that $U_{i,D} \leq D$ for all $D \in \mathcal{D}_\rho$ and all machines $i \in M$, and hence the vector $\hat{\mathrm{y}}$ satisfies the relaxed dbf-constraints (4.1) and (4.2) up to a factor 4. Also, Lemma 4.3.1 directly implies that the task assignment given by the vector $\hat{\mathrm{y}}$ is feasible with a speedup of 24 by choosing $\rho = 2$. However, using the definition of $U_{i,D}$ and a more careful calculation, the speedup can be bounded even further.

Theorem 4.3.4. *There is a $(11 + 4\sqrt{3})$-approximation test for the problem of assigning tasks to unrelated machines.*

Proof. In phase k of the iterative rounding algorithm, first, an extreme point solution y^k to the linear program LP^k is derived in polynomial time using for instance the simplex algorithm. Then, the algorithm fixes all variables $y_{i,\tau}^k$ which take on an integer value. Afterwards, it detects in polynomial time a constraint of the type (4.4b) or (4.4c) to which at most three fractional variables $y_{i,\tau}^k$ are assigned (Cases (i) and (ii) respectively). If so, then the corresponding constraint is removed from the linear program LP^k. If no such constraint can be found, the algorithm moves to Case (iii) and the procedure given in the proof of Lemma 4.3.3 is applied. This procedure will return an integer solution $\hat{\mathrm{y}}$ which identifies a partition of the tasks to the machines. All these operations run in polynomial time and preserve that at most three fractional tasks are fully assigned to each machine i. The returned integer solution $\hat{\mathrm{y}}$ respects the inequalities (4.5) and (4.6).

Let $\mathcal{T}$ denote the assignment implied by the integral variables $\hat{\mathrm{y}}$. Correspondingly, $\mathcal{T}_i$ denotes the set of tasks that are assigned the machine i, that is $\mathcal{T}_i = \{\tau \in T : \hat{y}_{i,\tau} = 1\}$. Finally, I prove the Theorem by showing that $\mathrm{dbf}_{i,\mathcal{T}}(s) \leq (11 + 4\sqrt{3})s$ for all s and each machine i.

$$\mathrm{dbf}_{i,\mathcal{T}}(s) = \sum_{\tau \in \mathcal{T}_i:\, d_\tau \leq s} \left\lfloor \frac{s + p_\tau - d_\tau}{p_\tau} \right\rfloor c_{i,\tau}$$

$$
\begin{aligned}
&\leq \sum_{\tau\in\mathcal{T}_i:\, d_\tau\leq s}\left(s\frac{c_{i,\tau}}{p_\tau}+c_{i,\tau}\right)\leq \sum_{\tau\in\mathcal{T}_i:\, r(d_\tau)\leq r(s)}\left(s\frac{c_{i,\tau}}{p_\tau}+c_{i,\tau}\right)\\
&\leq s\sum_{\tau\in T}\frac{c_{i,\tau}}{p_\tau}\,\hat{y}_{i,\tau}+\sum_{\tau\in T:\, r(d_\tau)\leq r(s)} c_{i,\tau}\,\hat{y}_{i,\tau}=s\sum_{\tau\in T}u_{i,\tau}\hat{y}_{i,\tau}+\sum_{k=0}^{\log_\rho(r(s))}\sum_{\tau\in T:\, r(d_\tau)=\rho^k} c_{i,\tau}\,\hat{y}_{i,\tau}\\
&\overset{(4.5)}{\leq} 4s+\sum_{k=0}^{\log_\rho(r(s))}\sum_{\tau\in T:\, r(d_\tau)=\rho^k} c_{i,\tau}\,\hat{y}_{i,\tau}\overset{(4.6)}{\leq} 4s+\sum_{k=0}^{\log_\rho(r(s))}\left(U_{i,\rho^k}+3\rho^k\right)\\
&\leq 4s+\rho^{\log_\rho(r(s))}+\sum_{k=0}^{\log_\rho(r(s))}3\rho^k=4s+r(s)+3\sum_{k=0}^{\log_\rho(r(s))}\rho^k\\
&=4s+r(s)+3\cdot\frac{\rho^{\log_\rho(r(s))+1}-1}{\rho-1}\leq 4s+r(s)+3\rho\frac{r(s)}{\rho-1}\\
&\leq s\left(4+\rho+3\frac{\rho^2}{\rho-1}\right)=\left(11+4\sqrt{3}\right)\cdot s.
\end{aligned}
$$

The last inequality follows since $r(s)\leq\rho\cdot s$ whereas the last inequality follows by optimizing of the value of ρ, i. e., set $\rho:=1+\sqrt{3}/2$. □

4.3.2 An improved approximation algorithm

In this subsection, I provide an improved algorithm which finds a solution violating the right-hand side of the constraints (4.3b) and (4.3c) by a lower factor than the factor of 4 by which they were violated in the previous subsection. In particular, I show the following lemma:

Lemma 4.3.5. *If the linear program ASS-LP2 given in* (4.4) *is feasible, then there exists an integer solution $\hat{\mathbf{y}}$ which satisfies:*

$$\sum_{i\in M}\hat{y}_{i,\tau}=1 \qquad \forall\tau\in T \tag{4.9a}$$

$$\sum_{\tau\in T}u_{i,\tau}\hat{y}_{i,\tau}\leq 3 \qquad \forall i\in M \tag{4.9b}$$

$$\sum_{\tau\in T:\, r(d_\tau)\leq D} c_{i,\tau}\hat{y}_{i,\tau}\leq U_{i,D}+2D \qquad \forall D\in\mathcal{D}_\rho,\,\forall i\in M \tag{4.9c}$$

$$\hat{y}_{i,\tau}\geq 0 \qquad \forall\tau\in T,\forall i\in M: u_{i,\tau}\leq 1\wedge c_{i,\tau}\leq d_\tau \tag{4.9d}$$

Note that condition (4.9d) explicitly states that, conform Properties 4.2.4 and 4.2.5, a variable $y_{i,\tau}$ is only created for the original linear program ASS-LP if $u_{i,\tau}\leq 1$ and $c_{i,\tau}\leq d_\tau$.

From Lemma 4.3.5 the main theorem of this subsection follows:

Theorem 4.3.6. *There is a $(8+6\sqrt{2/3})$-approximation test for the problem of assigning tasks to unrelated machines.*

I omit the proof of theorem since it is identical to the proof of Theorem 4.3.6. The only difference is given by the optimal value of ρ which now amounts to $1+\sqrt{3/2}$ yielding a total

speedup factor of $8+6\sqrt{2/3} \approx 12.89$. It remains to show Lemma 4.3.5. Part of the proof of this lemma resembles the proof of Theorem 3 of Karp et al. [74].

Proof of Lemma 4.3.5. I adjust ASS-LP2 given in (4.4) by normalizing the constraint (4.4c). That is, I divide the coefficients $c_{i,D}$ and $U_{i,D}$ by D. From now on I assume that ASS-LP2 refers to its normalized version. The gain is that the coefficients now satisfy $c_{i,D} \leq 1$.

I iteratively solve the linear program ASS-LP2 or a smaller version of it in case some variables and/or constraints have already been deleted. Again, let $\mathbf{y}^k$ be a feasible solution to the linear program LP^k which is under consideration in phase k of the algorithm. Let s be the number of variables present in that linear program. Let z_a, z_b and z_c be the number of constraints of type (4.4a), (4.4b), and (4.4c) respectively. Also, let $z = z_a + z_b + z_c$. In the procedure given below, either a variable is deleted in case $s > z$, or, if $s \leq z$, a constraint is deleted while ensuring that such a constraint in the final solution will not be violated too much. Along the way, the algorithm preserves the validity of constraint (4.4a).

Iteration k is initiated by some preprocessing. First, all variables $y^k_{i,\tau}$ which are already integer are fixed. Consequently, these variables are removed from the program and the right-hand side of the LP^k will be adjusted. Note that if there is some variable $y^k_{i,\tau} = 1$, not only this variable will be removed from the program, but also all remaining variables $y^k_{i',\tau}$ for $i' \neq i$ will be removed as those will equal zero. That this is a legitimate action follows from the corresponding constraint of type (4.4a) being satisfied at all times. Finally, if $y^k_{i,\tau} = 1$, then also the corresponding constraint of type (4.4a) is removed. Similarly, actually, any constraint left without any variable will be removed from the program.

In the remainder of the analysis I consider two cases: either $s > z$ or $s \leq z$. In the former case a variable will be removed whereas in the latter a constraint will be removed.

First, consider the case where $s > z$. Then, the number of remaining variables is larger than the number of remaining constraints. Hence, the nullspace of the matrix associated with the left-hand side of the constraints is nonempty. Therefore, an extreme point solution to LP^k can be found in polynomial time. Such an extreme point has at least one variable attaining an integral value (zero or one). This variable will be fixed and removed from the program decreasing the number of columns of the linear program by one. Along the way the condition (4.4a) is preserved. A more elaborate description of this procedure is given below and mimics the first part of the proof of Theorem 3 of Karp et al. [74].

The s variables are divided into $\max\{v, z+1\}$ groups where v is the number of distinct values all the variables take. Divide the variables in such a way, such that all variables in a single group have the same value. Now create a new feasible linear program where each new variable represents a group of variables in the original linear program LP^k. In particular, if the original linear program $(4.4)^k$ is denoted by $\mathbf{A}\mathbf{y} \leq \mathbf{b}$, then the new linear program $\mathbf{G}\mathbf{x} \leq \mathbf{h}$ is such that $\mathbf{h} := \mathbf{b}$, x_j equals the value the variables y_q in group j take (where q represents some machine-task-pair (i,τ)), and $g_{lj} := \sum_{q:q \text{ in group } j} a_{lq}$ for constraint l. Next, reduce $\mathbf{G}$ to row echelon form. This operation does not change the nullspace of $\mathbf{G}$. Let $\mathbf{x_0}$ be a vector in the nullspace of $\mathbf{G}$, define $\lambda^* := \min\{\lambda \geq 0 : \mathbf{x} + \lambda\mathbf{x_0} \text{ has an integral component}\}$ and let $\mathbf{w} := \mathbf{x} + \lambda^*\mathbf{x_0}$. Finally, an extreme point solution $\mathbf{w}$ is derived where some w_j is integral. For some original variable y_q in group j, fix its value and remove it from the program: delete the corresponding column in the linear program and adjust the right-hand side of the program. Start the next iteration of the algorithm.

Since only one variable at a time is removed, it cannot occur that a task will be deleted without being assigned to a machine. To be more explicit, if the variable $y_{i,\tau} = 0$ is removed, then at least one more variable $y_{i',\tau}$, where $i' \neq i$, remains due to preservation of the constraint (4.4a). One exception to removing one variable at a time is when the algorithm would like to remove $y_{i,\tau} = 1$. In that case, the algorithm also removes all remaining variables $y_{i',\tau}$ (which equal zero) for all $i' \neq i$, and remove the corresponding constraint (4.4a).

Secondly, consider the case where $s \leq z$. Let the matrix $\mathbf{A}$ represents the left-hand side coefficients of the linear program (4.4)^k. I show that if $s \leq z$, then there always exists a constraint l of type (4.4b) or (4.4c) such that

$$\max_{\mathbf{z} \in S} \left\{ (\mathbf{Az})_l - (\mathbf{Ay}^k)_l \right\} \leq 2. \tag{4.10}$$

Here, S denotes the integer solution space corresponding to all the remaining variables, that is, $S := \{0,1\}^s$ (each remaining $y^k_{i,\tau}$ will in the end either be 0 or 1). Knowing that such a constraint exists, it can be detected in polynomial time and removed from the program. The final solution (task partition) will then only violate the right-hand side of the deleted constraint up to a factor 3, satisfying the LP given in (4.9).

I show the claim given by (4.10) by contradiction. Assume no constraint l of type (4.4b) or (4.4c) exists such that $\max_{\mathbf{z} \in S} \left\{ (\mathbf{Az})_l - (\mathbf{Ay}^k)_l \right\} \leq 2$. That is, for each constraint l of type (4.4b) or (4.4c) it holds that there exists a vector $\mathbf{z} \in S$ such that

$$(\mathbf{Az})_l - (\mathbf{Ay}^k)_l > 2. \tag{4.11}$$

Recall that q represents a machine-task-pair (i, τ). A variable y^k_q (so $y^k_{i,\tau}$) can only be present in the program if it is not integer yet. Therefore, the corresponding task τ is not assigned yet to a single machine (but partially to several machines among which machine i). Consequently, the constraint of type (4.4a) corresponding to task τ is still present in the program. It follows that

$$\sum_q y^k_q = \sum_{i \in M} \sum_{\tau : 0 < y^k_{i,\tau} < 1} y^k_{i,\tau} = z_a, \tag{4.12}$$

where z_a is the number of remaining constraints of type (4.4a), that is, the number of tasks remaining which are not yet fully assigned to a single machine. In order to proceed I need a few more definitions. Let L refer to the set of constraints of type (4.4b) and (4.4c) which are still present in the linear program LP^k. In particular, let L^q denote the set of constraints of type (4.4b) and (4.4c) still present in the linear program LP^k in which the variable y_q is present. I derive the following chain of inequalities:

$$\begin{aligned}
2(z - z_a) &= 2(z_b + z_c) \\
&\overset{(4.11)}{<} \sum_{l \in L} \max_{\mathbf{z} \in S} \left((\mathbf{Az})_l - (\mathbf{Ay}^k)_l \right) \overset{\text{as all } a_{lq} \geq 0}{=} \sum_{l \in L} \left((\mathbf{A1})_l - (\mathbf{Ay}^k)_l \right) \\
&= \sum_{l \in L} \sum_q a_{lq} \left(1 - y^k_q \right) = \sum_q \sum_{l \in L^q} a_{lq} \left(1 - y^k_q \right) \\
&\leq \sum_q 2 \left(1 - y^k_q \right) = 2s - \sum_q 2 y^k_q \overset{(4.12)}{=} 2s - 2z_a.
\end{aligned} \tag{4.13}$$

Here, $\mathbf{1}$ represents the vector of all ones. The last inequality follows since each variable y_q^k appears in at most one constraint of type (4.4b) and in at most one constraint of type (4.4c). Also, it was used in the last inequality that $u_{i,\tau} \leq 1$ and $c_{i,\tau} \leq 1$ for all machines i and tasks τ, that is, $a_{lq} \leq 1$ for all remaining variables q and all remaining constraints $l \in L^q$. Therefore, $\sum_{l \in L^q} a_{lq} \leq 2$. The chain of inequalities implies that $2(z - z_a) < 2(s - z_a)$, i. e., $z < s$, which is a contradiction to being in the case where $s \leq z$.

In conclusion, I showed that if $s \leq z$, then there must be a constraint $l \in L$ for which $\max_{\mathbf{z} \in I}(\mathbf{Az})_l - (\mathbf{A}\mathbf{y}^k)_l \leq 2$. This constraint l can then be removed from the program. Note that that such a constraint can be detected in polynomial time by simply checking for each constraint l of type (4.4b) or (4.4c) whether $\sum_q (1 - y_q^k) \leq 2$. Also, this condition is sufficient since all $a_{lq} \geq 0$ and the maximum value any variable y_q^k can take is 1. If constraint l is of type (4.4b) or (4.4c) respectively, then in the final solution (partition) this constraint will satisfy (4.9b) or (4.9c) respectively. □

4.3.3 $2 - \epsilon$ hardness result

To complete this section, I show that it is *$\mathcal{NP}$-hard* to decide whether a task system T has a feasible partition on m unrelated machines, even with a speedup factor of $2 - \epsilon$for any $\epsilon > 0$. The proof follows the lines of the $(\frac{3}{2} - \epsilon)$-hardness result for makespan minimization in [78].

Theorem 4.3.7. *Let $\epsilon > 0$. There is no $(2 - \epsilon)$-approximation test for the problem of assigning tasks to unrelated machines, unless $\mathcal{P} = \mathcal{NP}$.*

Proof. I show that, unless $\mathcal{P} = \mathcal{NP}$, it is not possible to decide in polynomial time whether an instance of sporadic realtime scheduling problem has an assignment which is feasible on m unrelated machines, even under a speedup factor up to 2. The reduction is from the 3-Dimensional Matching problem which is known to be *$\mathcal{NP}$-complete* [78] and introduced below.

3-DIMENSIONAL MATCHING: Given three disjoint sets each consisting of n elements: $A = \{a_1, a_2, \ldots, a_n\}$, $B = \{b_1, b_2, \ldots, b_n\}$, $C = \{c_1, c_2, \ldots, c_n\}$, and consider a family $F = \{R_1, R_2, \ldots, R_m\}$ of $m \geq n$ triples such that each triple contains exactly one element from the set A, one from B, and one from C, that is, $|R_i \cap A| = |R_i \cap B| = |R_i \cap C| = 1$ for all $i \in \{1, \ldots, m\}$. The decision question is the following: *Does F contain a 3-dimensional matching, i. e., a subfamily F' for which $|F'| = n$ and $\bigcup_{R_i \in F'} R_i = A \cup B \cup C$?*

Let $\epsilon > 0$ and let $\rho = 2 - \epsilon$. Consider an instance $\mathcal{I}$ of 3-Dimensional matching. Create an instance (task system) T for sporadic realtime scheduling on unrelated parallel machines in the following way:

- Define a large constant $M := 2/\epsilon$.
- Associate a machine i with each triple R_i, yielding m machines.
- Let $R(a_k) \subseteq R$ be the set of triples containing element $a_k \in A$, and let $r(a_k) = |R(a_k)|$. Note that $\sum_{a_k \in A} r(a_k) = m$.
- For each element $b_h \in B$, create one task τ_h such that $d_{\tau_h} = 1$ and $p_{\tau_h} = \infty$. Further, c_{i,τ_h} equals 1 if $b_h \in R_i$, and equals 2 otherwise. I refer to these task as being type B tasks. There will be n tasks of type B.

- For each element $c_l \in C$, create one task τ_l such that $d_{\tau_l} = M$ and $p_{\tau_l} = \infty$. Further, c_{i,τ_l} equals $M - 1$ if $c_l \in R_i$, and equals $2M$ otherwise. I refer to these task as being type C tasks. There will be n tasks of type C.

- For each element $a_k \in A$, create $r(a_k) - 1$ dummy tasks $dum(k_1), \ldots, dum(k_{r(a_k)-1})$ which have a deadline and period equal to one. Each dummy task $dum(k_z)$ has $c_{i,k_z} = 1$ if $i \in R(a_k)$, and $c_{i,k_z} = 2$ otherwise. Note that in total there will be $m - n$ dummy tasks.

Lemma 4.3.8. *If there exists a 3-dimensional matching for $\mathcal{I}$, then there exists a feasible assignment for T.*

Proof. Let $R^* \subseteq R$ be the triples in the 3-dimensional matching. For all triples $R_i = \{a_k, b_h, c_l\} \in R^*$, schedule tasks τ_h and τ_l on machine i. Note that machine i can process the tasks assigned to it. Also, all tasks corresponding to elements in B and C have been scheduled. As the element a_k is only covered by one triple in R^*, it follows that there are $r(a_k) - 1$ machines remaining in $R(a_k)$ which are not assigned any tasks yet. Assign the $r(a_k) - 1$ dummy tasks $dum(k_1), \ldots dum(k_{r(a_k)-1})$ to these machines. Note that a machine in $R(a_k)$ can process exactly one dummy task corresponding to the element a_k. □

Lemma 4.3.9. *For any $\rho < 2$, if there exists a feasible assignment for T with speedup ρ, then there exists a 3-dimensional matching for $\mathcal{I}$.*

Proof. Any assignment of T which is feasible under a speedup of ρ satisfies the following properties:

Proposition 4.3.10. *For any $\rho < 2$, no task τ_h corresponding to an element $b_h \in B$ can be scheduled to a machine i if $b_h \notin R_i$, even under a speedup of ρ.*

Proposition 4.3.11. *For any $\rho < 2$, no task τ_l corresponding to an element $c_l \in C$ can be scheduled to a machine i if $c_l \notin R_i$, even under a speedup of ρ.*

Proposition 4.3.12. *For any $\rho < 2$, no dummy task $dum(k_z)$ corresponding to an element $a_k \in A$ can be scheduled to a machine i if $i \notin R(a_k)$, even under a speedup of ρ.*

I show Proposition 4.3.10. Propositions 4.3.11 and 4.3.12 follow similarly.

Proof. By contradiction, let a task τ_h corresponding to an element $b_h \in B$ be scheduled to a machine i such that $b_h \notin R_i$. Then $c_{i,\tau_h} = 2$. At time $d_{\tau_h} = 1$ the first job of task τ_h needs to be completed and hence speedup of 2 is required. □

Proposition 4.3.13. *For any $\rho < 2$, no dummy task $dum(k_z)$ can be scheduled together with another task on the same machine, even under a speedup of ρ.*

Proof. The proof is by contradiction. Consider the following three cases:

- Two dummy tasks are scheduled on the same machine. Both dummies need to finish their first job by their first deadline which is at time 1. Their accumulated processing requirement to the machine is at least 2 and hence a speedup of at least 2 is required.

- A dummy task and a task of type B are scheduled on the same machine. Reasoning analogously to the previous case.

- A dummy task and a task of type C are scheduled on the same machine. Consider time moment M, then the dummy task needs to have finished M jobs whereas the other task needs to have finished its first job. The accumulated processing requirement the machine needs to have processed by time M is at least $M * 1 + (M - 1) = 2M - 1$. Hence a speedup of $\frac{2M-1}{M} = 2 - \frac{1}{M} = 2 - \epsilon/2 > \rho$ is required, by my definition of M.

□

Proposition 4.3.13 yields that each dummy task is scheduled on its own machine, even under a speedup of $\rho < 2$. Therefore, and by Proposition 4.3.12, for all $a_k \in A$, there remains in each group $R(a_k)$ one machine available to process tasks of type B or C, even under a speedup of $\rho < 2$. In total there are $m - (m - n) = n$ machines left which do not process a dummy task. There are $2n$ tasks of type B and C. Since, a single machine cannot process two tasks of the same type under a speedup less than 2, it follows that each machine which does not process a dummy task, processes one task of type B and one task of type C. Let $i_k \in R(a_k)$ be the machine which does not process a dummy task but instead one task of type B and one task of type C. By Propositions 4.3.10 and 4.3.11, the only way for machine i_k to be feasible, even under a speedup of $\rho < 2$, is when it processes the tasks corresponding to b_h and c_l where $R_{i_k} = \{a_k, b_h, c_l\}$. It follows that the machines i_k, for $k \in \{1, \ldots, m\}$, define a 3-dimensional matching for $\mathcal{I}$. □

Theorem 4.3.7 follows by Lemmas 4.3.8 and 4.3.9, and by the 3-DIMENSIONAL MATCHING problem being $\mathcal{NP}$*-complete*. □

4.4 Constant number of machines

Assuming that the number of machines m is bounded by a constant, this section presents a dynamic programming algorithm (DP) that gives a $(1 + \epsilon)$-approximation test for any $\epsilon > 0$. For having a DP-table of bounded size I introduce an approximation of the demand bound function such that the contribution of each task can be derived by using only a constant number of values.

Recall that it is assumed that $d_{\min} = \min_{\tau \in T} d_\tau = 1$. Let $\epsilon > 0$, and assume without loss of generality that $\epsilon < 1/2$. Let L be the minimum integer which satisfies $1 \leq (1 + \epsilon)^{L-1}\epsilon^2$. I define the function dbf^*:

$$\mathrm{dbf}^*_{i,\tau}(s) := \begin{cases} \left\lfloor \frac{s+p_\tau-d_\tau}{p_\tau} \right\rfloor c_{i,\tau} & \text{if } s < (1+\epsilon)^L \cdot d_\tau \\ \frac{c_{i,\tau}}{p_\tau} s & \text{otherwise.} \end{cases} \tag{4.14}$$

Given a task assignment $\mathcal{T}$ of tasks in T to the machines, I define

$$\mathrm{dbf}^*_{i,\mathcal{T}}(s) := \sum_{\tau \in \mathcal{T}_i} \mathrm{dbf}^*_{i,\tau}(s) \quad \forall s > 0.$$

Further, for ease of notation, I write $\mathrm{dbf}^*_i(s)$ instead of $\mathrm{dbf}^*_{i,\mathcal{T}}(s)$ in case the assignment $\mathcal{T}$ is clear from the context. The key observation is that for computing the function $\mathrm{dbf}^*_{i,\tau}(s)$ for a fixed task τ, it suffices to know the utilization of the task τ and the values of the demand

bound function $\text{dbf}_{i,\tau}(s)$ for $s \in \left[d_\tau, (1+\epsilon)^L \cdot d_\tau\right)$. Exploiting the properties of the functions $\text{dbf}_{i,\tau}(s)$ and $\text{dbf}^*_{i,\tau}(s)$ yields that $\text{dbf}^*_{i,\mathcal{T}}$ is a $(1+\epsilon)$-approximation of the 'real' demand bound function $\text{dbf}_{i,\mathcal{T}}$.

Lemma 4.4.1. *Given an assignment $\mathcal{T}$ and a constant $\epsilon < 1/2$. Then, for all machines i,*
(i) if $dbf^*_{i,\mathcal{T}}(r) \leq \alpha \cdot r$ for all $r \geq 0$, then $dbf_{i,\mathcal{T}}(s) \leq (1+\epsilon) \cdot \alpha \cdot s$ for all $s \geq 0$;
(ii) if $dbf_{i,\mathcal{T}}(r) \leq r$ for all $r \geq 0$, then $dbf^*_{i,\mathcal{T}}(s) \leq (1+\epsilon) \cdot s$ for all $s \geq 0$.

Proof. The first claim trivially holds for all $s \leq d_{\min}$ as then $\text{dbf}_{i,\tau}(s) = 0$ for all tasks $\tau \in T$. Assume by induction that claim (i) holds true for all $s' < s$, I show that (i) then also holds for s. Consider some partition of the tasks $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_m\}$. Let $\mathcal{T}_i^{\text{early}} := \{\tau \in \mathcal{T}_i | (1+\epsilon)^L \cdot d_\tau < s\}$ and $\mathcal{T}_i^{\text{late}} := \mathcal{T}_i \backslash \mathcal{T}_i^{\text{early}}$. By definition of $\text{dbf}^*_{i,\mathcal{T}}$ it follows that

$$\text{dbf}_{i,\mathcal{T}}(s) \leq \sum_{\tau \in \mathcal{T}_i} \text{dbf}^*_{i,\tau}(s) + \sum_{\tau \in \mathcal{T}_i^{\text{early}}} c_{i,\tau} = \text{dbf}^*_{i,\mathcal{T}}(s) + \sum_{\tau \in \mathcal{T}_i^{\text{early}}} c_{i,\tau}.$$

Further, by definition of $\text{dbf}^*_{i,\mathcal{T}}$ and $\mathcal{T}_i^{\text{early}}$ it is that

$$\begin{aligned}
&\sum_{\tau \in \mathcal{T}_i^{\text{early}}} c_{i,\tau} \leq \sum_{\tau \in \mathcal{T}_i^{\text{early}}} \left\lfloor \frac{\frac{s}{(1+\epsilon)^L} + p_\tau - d_\tau}{p_\tau} \right\rfloor c_{i,\tau} \\
&= \sum_{\tau \in \mathcal{T}_i^{\text{early}}} \text{dbf}_{i,\tau}\left(\frac{s}{(1+\epsilon)^L}\right) = \text{dbf}_{i,\mathcal{T}}\left(\frac{s}{(1+\epsilon)^L}\right) \leq \frac{\alpha s}{(1+\epsilon)^{L-1}} \leq \alpha s \epsilon^2 \leq \alpha s \epsilon.
\end{aligned}$$

Here, the second last inequality follows from $\frac{1}{(1+\epsilon)^{L-1}} < \epsilon^2$ and the induction step. The two above inequalities imply that $\text{dbf}_{i,\mathcal{T}}(s) \leq \text{dbf}^*_{i,\mathcal{T}}(s) + \alpha s \epsilon \leq \alpha s (1+\epsilon)$. Regarding the second claim, similar reasoning shows that

$$\begin{aligned}
\text{dbf}^*_{i,\mathcal{T}}(s) &< \sum_{\tau \in \mathcal{T}_i^{\text{late}}} \text{dbf}_{i,\tau}(s) + \sum_{\tau \in \mathcal{T}_i^{\text{early}}} \frac{s - d_\tau + \frac{s}{(1+\epsilon)^L}}{p_\tau} c_{i,\tau} \\
&\leq \sum_{\tau \in \mathcal{T}_i^{\text{late}}} \text{dbf}_{i,\tau}(s) + \sum_{\tau \in \mathcal{T}_i^{\text{early}}} \left(\left\lfloor \frac{s - d_\tau + p_\tau}{p_\tau} \right\rfloor + \frac{s}{(1+\epsilon)^L p_\tau} \right) c_{i,\tau} \\
&\leq \text{dbf}_{i,\mathcal{T}}(s) + \sum_{\tau \in \mathcal{T}_i^{\text{early}}} (\epsilon^2 \cdot s \cdot u_{i,\tau}) = \text{dbf}_{i,\mathcal{T}}(s) + \epsilon^2 \text{dbf}^*_{i,\mathcal{T}_i^{\text{early}}}(s) \\
&\leq \text{dbf}_{i,\mathcal{T}}(s) + \epsilon^2 \text{dbf}^*_{i,\mathcal{T}}(s)
\end{aligned}$$

Therefore, $(1-\epsilon^2)\text{dbf}^*_{i,\mathcal{T}}(s) \leq \text{dbf}_{i,\mathcal{T}}(s)$, i. e., $\text{dbf}^*_{i,\mathcal{T}}(s) \leq (1+\epsilon)\text{dbf}_{i,\mathcal{T}}(s)$, for any $\epsilon < 1/2$. □

Note that L has been defined in such a way to facilitate the above proof. Further, observe that in contrast to other approximations of the demand bound function considered in the literature (e. g., [2]), in Lemma 4.4.1, I do not use task by task analysis, and I do not bound the ratio $\text{dbf}_\tau(s)/\text{dbf}^*_\tau(s)$. In fact, the latter can be unbounded: consider for example a task τ with $c_\tau = 1$, $d_\tau = 1$, and $p_\tau = M$ for a very large value M, then $\text{dbf}_\tau(s) \geq 1$ for all $s \geq 1$ whereas $\text{dbf}^*_\tau((1+\epsilon)^L) = (1+\epsilon)^L/M$.

The following lemma shows that at the cost of a $(1+\epsilon)$-speedup it suffices to check whether the condition $\mathrm{dbf}^*_{i,\mathcal{T}}(s) \leq s$ is (approximately) satisfied at powers of $1+\epsilon$. Therefore, the DP may characterize each task τ only by its utilization and the constantly many values $\mathrm{dbf}^*_{i,\tau}((1+\epsilon)^k)$ (namely those values for integers k such that $d_\tau \leq (1+\epsilon)^k < (1+\epsilon)^L \cdot d_\tau$, for each machine i).

Lemma 4.4.2. *Consider a task τ and a machine $i \in M$. If for all $k \in \mathbb{N}_{\geq 0}$ it holds that $\mathrm{dbf}^*_{i,\tau}((1+\epsilon)^k) \leq \alpha \cdot (1+\epsilon)^k$ then $\mathrm{dbf}^*_{i,\tau}(s) \leq \alpha \cdot s \cdot (1+2\epsilon)$ for all $s \geq 0$.*

Proof. Assume that $\mathrm{dbf}^*_{i,\tau}((1+\epsilon)^k) \leq \alpha \cdot (1+\epsilon)^k$ for all $k \in \mathbb{N}_{\geq 0}$. I need to show that

$$\mathrm{dbf}^*_{i,\tau}(s) \leq \alpha \cdot s \cdot (1+2\epsilon) \quad \forall s \geq 0. \tag{4.15}$$

The claim (4.15) trivially holds for all $0 \leq s < 1$ as for such s it holds that $\mathrm{dbf}^*_{i,\tau}(s) = 0$ since $d_{\min} = 1$. Define the integer k^* to be the smallest integer such that $(1+\epsilon)^{k^*} \geq (1+\epsilon)^L \cdot d_\tau$. Note that $k^* \geq L$ as $d_{\min} \geq 1$. Further, for $s \leq (1+\epsilon)^{k^*-1}$ it holds that $\mathrm{dbf}^*_{i,\tau}(s) = \left\lfloor \frac{s+p_\tau-d_\tau}{p_\tau} \right\rfloor c_{i,\tau}$, whereas for $s \geq (1+\epsilon)^{k^*}$ it holds that $\mathrm{dbf}^*_{i,\tau}(s) = u_{i,\tau} \cdot s$. Then, on the range

$$s \in \left[1;\, (1+\epsilon)^{k^*-1}\right) \cup \left[(1+\epsilon)^{k^*};\, \infty\right)$$

the function $\mathrm{dbf}^*_{i,\tau}(s)$ is, by definition, nondecreasing. Therefore, for such s the following chain of inequalities holds:

$$\mathrm{dbf}^*_{i,\tau}(s) \leq \mathrm{dbf}^*_{i,\tau}\left((1+\epsilon)^{\lceil \log_{1+\epsilon} s \rceil}\right) \leq \alpha(1+\epsilon)^{\lceil \log_{1+\epsilon} s \rceil} \leq \alpha s(1+\epsilon),$$

showing claim (4.15). The function $\mathrm{dbf}^*_{i,\tau}(s)$ changes form exactly at the point $s = (1+\epsilon)^L d_\tau$. Therefore, at this point only, the function might actually be decreasing such that the above chain of inequalities can not be used for any s in the range $\left[(1+\epsilon)^{k^*-1};\, (1+\epsilon)^{k^*}\right)$. Consequently, for such s, some more detailed work is needed to prove (4.15). Define $s^- = (1+\epsilon)^{k^*-1}$, then

$$\begin{aligned}
\mathrm{dbf}^*_{i,\tau}(s) &= \left\lfloor \frac{s+p_\tau-d_\tau}{p_\tau} \right\rfloor c_{i,\tau} < \left\lfloor \frac{(1+\epsilon)s^- + p_\tau - d_\tau}{p_\tau} \right\rfloor c_{i,\tau} \\
&\leq \frac{(1+\epsilon)s^- + p_\tau - d_\tau}{p_\tau} c_{i,\tau} = \frac{\epsilon s^-}{p_\tau} c_{i,\tau} + \frac{s^- + p_\tau - d_\tau}{p_\tau} c_{i,\tau} \\
&\leq \frac{\epsilon s^-}{p_\tau} c_{i,\tau} + c_{i,\tau} + \left\lfloor \frac{s^- + p_\tau - d_\tau}{p_\tau} \right\rfloor c_{i,\tau} = \epsilon s^- \cdot u_{i,\tau} + c_{i,\tau} + \mathrm{dbf}^*_{i,\tau}(s^-) \\
&\leq \epsilon s^- \cdot u_{i,\tau} + c_{i,\tau} + \alpha s^- \leq \epsilon s^- \cdot u_{i,\tau} + c_{i,\tau} + \alpha s.
\end{aligned} \tag{4.16}$$

Further,

$$s^- \cdot u_{i,\tau} = (1+\epsilon)^{k^*-1} \cdot u_{i,\tau} = \frac{\mathrm{dbf}^*_{i,\tau}((1+\epsilon)^{k^*})}{1+\epsilon} \leq \alpha(1+\epsilon)^{k^*-1} = \alpha s^-. \tag{4.17}$$

Finally, I upper bound $c_{i,\tau}$ using that, by definition of k^*, $s^- = (1+\epsilon)^{k^*-1} \geq (1+\epsilon)^{L-1} \cdot d_\tau$:

$$c_{i,\tau} \leq \left\lfloor \frac{\frac{s^-}{(1+\epsilon)^{L-1}} + p_\tau - d_\tau}{p_\tau} \right\rfloor c_{i,\tau} = \mathrm{dbf}^*_{i,\tau}\left(\frac{s^-}{(1+\epsilon)^{L-1}}\right) \leq \frac{\alpha \cdot s^-}{(1+\epsilon)^{L-1}} \leq \alpha \cdot \epsilon \cdot s^-, \tag{4.18}$$

where the last inequality follows from the definition of L and since $\epsilon^2 < \epsilon$. Integrating the inequalities (4.16), (4.17) and (4.18) shows claim (4.15), that is,

$$\mathrm{dbf}^*_{i,\tau}(s) < \epsilon u_{i,\tau} \cdot s^- + c_{i,\tau} + \alpha s \leq 2\alpha \cdot \epsilon \cdot s^- + \alpha s \leq (1+2\epsilon) \cdot \alpha \cdot s.$$

□

For each task τ, each machine i and all $\ell \in \mathbb{N}_{\geq 0}$, I introduce a vector $v(i,\tau)$ by defining position $v(i,\tau)_\ell := \mathrm{dbf}^*_{i,\tau}((1+\epsilon)^\ell)/(1+\epsilon)^\ell$. Recall that $\|\mathbf{a}\|_\infty = \max_i \{a_i\}$ for any vector $\mathbf{a}$. The following property follows by definition.

Property 4.4.3. *Consider an assignment $\mathcal{T}$. For all machines $i \in M$, it holds true that $\left\|\sum_{\tau\in\mathcal{T}_i} v(i,\tau)\right\|_\infty \leq \alpha$ if and only if $\mathrm{dbf}^*_{i,\mathcal{T}}((1+\epsilon)^\ell) \leq \alpha(1+\epsilon)^\ell$, for all $\ell \in \mathbb{N}_{\geq 0}$.*

I present a dynamic programming algorithm which either (i) asserts that there is no feasible assignment of the tasks to the machines by showing that there is no assignment $\mathcal{T}$ of tasks to machines such that $\left\|\sum_{\tau\in\mathcal{T}_i} v(i,\tau)\right\|_\infty \leq 1$ for each machine i, or (ii) finds an assignment $\mathcal{T}$ such that $\left\|\sum_{\tau\in\mathcal{T}_i} v(i,\tau)\right\|_\infty \leq 1 + O(\epsilon)$ for each machine i. In the latter case, Lemmas 4.4.1 and 4.4.2, and the above property imply an approximation test for the problem of assigning tasks to a constant number of unrelated machines. The test either concludes that the task system is not feasible (without speedup) or provides an assignment which is feasible in case the machines have a speedup factor of $1 + O(\epsilon)$.

Assume without loss of generality that the tasks $\tau_1, \ldots, \tau_n$ are ordered such that $d_{\tau_q} \leq d_{\tau_{q+1}}$ for each q. I partition the tasks into groups $G_k := \{\tau | (1+\epsilon)^k \leq d_\tau < (1+\epsilon)^{k+1}\}$ for each $k \in \mathbb{N}_{\geq 0}$. The proposed DP works in phases; one phase for each task. The key idea is that when trying to assign task $\tau \in G_k$, the DP needs only a constant number of values from the assignment of the previously considered tasks. With $L^{(k)} := \min\{k, L\}$, for each machine i the DP needs

- the sum $\sum_{\tau\in\mathcal{T}_i\cap\left(\bigcup_{k'=0}^{k-L^{(k)}} G_{k'}\right)} u_{i,\tau}$,
- the sum $\sum_{\tau\in\mathcal{T}_i\cap G_{k'}} u_{i,\tau}$, for all $k' :\ k - L^{(k)} < k' \leq k$, and
- the sum $\sum_{\tau\in\mathcal{T}_i\cap G_{k'}} v(i,\tau)_\ell$ for all $\ell :\ k \leq \ell \leq k + L$ and all $k' : \ell - L^{(\ell)} < k' \leq k$.

Ideally, I would like the DP to store all possible combinations of the above quantities that can result from assigning the tasks of previous iterations. Then, the DP could compute the values for the next iteration by taking each combination of values from the last iteration and additionally schedule task τ to one of the machines. Unfortunately, the number of possible combinations of the above values is not polynomially bounded. In order to bound them, I round entries of the vectors $v(i,\tau)$. The DP then performs the described procedure with the rounded vectors. This will result in a polynomial time procedure.

I continue with formally presenting the dynamic programming algorithm. Consider a task $\tau \in G_k$, $k \in \mathbb{N}_{\geq 0}$. For each i, define $v'(i,\tau)_\ell := \frac{\epsilon}{n}\left\lfloor \frac{n}{\epsilon} \cdot v(i,\tau)_\ell \right\rfloor$ for each $\ell < k + L$, and $v'(i,\tau)_{\ell'} := u'_{i,\tau} := \frac{\epsilon}{n}\left\lfloor \frac{n}{\epsilon} \cdot u_{i,\tau} \right\rfloor$ for each $\ell' \geq k + L$. The following lemma bounds the rounding error.

Lemma 4.4.4. *Let i be a machine and $\mathcal{T}_i$ be a set of tasks. For all $\ell \in \mathbb{N}_{\geq 0}$, it holds that $\sum_{\tau\in\mathcal{T}_i} v'(i,\tau)_\ell \leq \sum_{\tau\in\mathcal{T}_i} v(i,\tau)_\ell \leq \epsilon + \sum_{\tau\in\mathcal{T}_i} v'(i,\tau)_\ell$.*

Proof. Consider a task $\tau \in \mathcal{T}_i$. Define $k(\tau)$ such that $\tau \in G_{k(\tau)}$. I show for each $\tau \in \mathcal{T}_i$ and the corresponding $k(\tau)$ that $v'(i,\tau)_\ell \leq v(i,\tau)_\ell \leq v'(i,\tau)_\ell + (\epsilon/n)$. From this, the statement trivially follows. The case where $\ell < k(\tau) + L$ follows trivially from the relation $\lfloor x \rfloor \leq x < \lfloor x \rfloor + 1$ which holds for any x. Therefore, consider the case $\ell \geq k(\tau) + L$. Since $\tau \in G_{k(\tau)}$ it follows that $d_\tau < (1+\epsilon)^{k(\tau)+1}$. Hence, $\mathrm{dbf}^*_{i,\tau}((1+\epsilon)^\ell) = \frac{c_{i,\tau}}{p_\tau}(1+\epsilon)^\ell = u_{i,\tau}(1+\epsilon)^\ell$ which yields $v(i,\tau)_\ell = u_{i,\tau}$. The result for the case $\ell \geq k(\tau) + L$ now also follows from $\lfloor x \rfloor \leq x < \lfloor x \rfloor + 1$. □

Note that now each rounded vector $v'(i,\tau)$ can be described with only constantly many pieces of information. When working with the rounded vectors, for the quantities mentioned above there are only a polynomial number of combinations (assuming that m is a constant). In particular, the dynamic programming table will be of polynomial size. Formally, the dynamic programming table consists of entries of the form $(q, \mathbf{z}, \mathbf{w}, \mathbf{c})$ where

- $q \in \{0, ..., n\}$ denotes the phase of the DP. In phase q, task τ_q is being assigned to a machine. Let k be an integer such that $\tau_q \in G_k$;
- for each machine i, the value z_i is of the form $\ell \cdot \frac{\epsilon}{n}$ for some integer ℓ, denoting the rounded aggregated utilization of machine i due to the tasks having a deadline at least a factor of $(1+\epsilon)^L$ smaller with respect to the deadline of task τ_q;
- for each machine i and each k' with $k - L^{(k)} < k' \leq k$, the value $w_{i,k'}$ is of the form $\ell \cdot \frac{\epsilon}{n}$ for some integer ℓ, denoting the rounded utilization of tasks in $G_{k'} \cap \mathcal{T}_i$.
- for each triple $(i, k', k'') \in C_q$ with $C_q = \{(i, k', k'') : 1 \leq i \leq m;\ k \leq k'' < k + L$ and $k'' - L^{(k'')} < k' \leq k\}$, the value $c_{i,k',k''}$ is of the form $\ell \cdot \frac{\epsilon}{n}$ for some integer ℓ, denoting the quantity $\sum_{\tau \in \mathcal{T}_i \cap G_{k'}} v'(i,\tau)_{k''}$. Intuitively, it expresses how much the vectors of the tasks in $G_{k'}$ on machine i contribute towards dimension k''.

I require the following set of conditions to be satisfied for a DP-cell $(q, \mathbf{z}, \mathbf{w}, \mathbf{c})$ to exist; for each machine $i \in M$ and all $k'' \in \{k, \ldots, k + L\}$

$$z_i + \sum_{k'=k-L^{(k)}+1}^{k''-L^{(k'')}} w_{i,k'} + \sum_{k'=k''-L^{(k'')}+1}^{k} c_{i,k',k''} \leq 1 + \epsilon \tag{4.19}$$

In particular, this condition implies that for all parameters, $z_i,\ w_{i,k'},\ c_{i,k',k''} \leq 1 + \epsilon$.

Property 4.4.5. *The number of DP-cells is bounded by* $n \cdot ((1+\epsilon)n/\epsilon)^{2m \cdot L^2}$.

Proof. The values z_i, $w_{i,k'}$ and $c_{i,k',k''}$ are all stored with accuracy $\frac{\epsilon}{n}$ and hence each of those can take $(1+\epsilon)n/\epsilon$ many different realizations. Further, i can take m different values whereas k' and k'' can take L different values if the DP is in a certain phase q. Moreover, there are at most n phases for the DP. It follows that the number of cells of the DP table is no more than

$$n * \left(\frac{(1+\epsilon)n}{\epsilon}\right)^m * \left(\frac{(1+\epsilon)n}{\epsilon}\right)^{mL} * \left(\frac{(1+\epsilon)n}{\epsilon}\right)^{mL^2} \leq n * \left(\frac{(1+\epsilon)n}{\epsilon}\right)^{2mL^2}.$$

□

Each entry $(q, \mathbf{z}, \mathbf{w}, \mathbf{c})$ of the DP-table either stores 'YES' or 'NO' depending on whether or not there is an assignment of the tasks $\tau_1, \ldots, \tau_q$ to the machines which yields the quantities given by the vectors $\mathbf{z}, \mathbf{w}, \mathbf{c}$.

I proceed with describing how the DP-table is filled. First, initialize the table by assigning a 'YES'-entry to $(0, \mathbf{0}, \mathbf{0}, \mathbf{0})$ and a 'NO'-entry to any other entry with $q = 0$. Assume that for some phase q, all entries of the form $(q-1, \mathbf{z}^{(q-1)}, \mathbf{w}^{(q-1)}, \mathbf{c}^{(q-1)})$ have been computed. The DP-table is iteratively extended for some phase q. Phase q considers each combination of assigning task τ_q to some machine i and a DP-cell $(q-1, \mathbf{z}^{(q-1)}, \mathbf{w}^{(q-1)}, \mathbf{c}^{(q-1)})$ with a 'YES'-entry. Intuitively, the DP computes which values for $\mathbf{z}^{(q)}$, $\mathbf{w}^{(q)}$, and $\mathbf{c}^{(q)}$ are obtained if it takes the task assignment encoded in the DP-cell $(q-1, \mathbf{z}^{(q-1)}, \mathbf{w}^{(q-1)}, \mathbf{c}^{(q-1)})$ and additionally schedules task τ_q to machine i.

Formally, let tasks τ_{q-1} and τ_q be in group G_h and G_k, respectively. Almost all entries of the vectors are equal and hence I only list the values which differ. If $h = k$, then $w_{i,k}^{(q)} = w_{i,k}^{(q-1)} + u'_{i,\tau}$, and $c_{i,k,k''}^{(q)} = c_{i,k,k''}^{(q-1)} + v'(i,\tau)_{k''}$ for all $k'' \in \{k, \ldots, k+L\}$. If $h \neq k$ it is safe to assume that $h = k-1$ by creating dummy tasks of zero processing requirement. Then, $z_g^{(q)} = z_g^{(q-1)} + w_{g,k-L^{(k)}}^{(q-1)}$ for all machines $g \in M$; $w_{g,k'}^{(q)} = w_{g,k'}^{(q-1)}$ for machines $g \in M$ and all $k' : k - L^{(k)} < k' < k$; $w_{i,k}^{(q)} = u'_{i,\tau_q}$ and $w_{g,k}^{(q)} = 0$ for all machines $g \neq i$; $c_{g,k',k''}^{(q)} = c_{g,k',k''}^{(q-1)}$ for all machines $g \in M$, all $k'' : k \leq k'' \leq k+L$ and all $k' : k'' - L^{(k'')} < k' < k$; $c_{i,k,k''}^{(q)} = v'(i,\tau)_{(k'')}$ for all $k'' : k \leq k'' \leq k+L$; and $c_{g,k,k''}^{(q)} = 0$ for all machines $g \neq i$ and all $k'' : k \leq k'' \leq k+L$.

Finally, the DP checks whether the computed values $\mathbf{z}^{(q)}, \mathbf{w}^{(q)}$ and $\mathbf{c}^{(q)}$ satisfy the condition given in (4.19). If this is the case, then the corresponding DP-cell $(q, \mathbf{z}^{(q)}, \mathbf{w}^{(q)}, \mathbf{c}^{(q)})$ is filled with a 'YES'-entry and I say that this DP-cell *extends* the DP-cell $(q-1, \mathbf{z}^{(q-1)}, \mathbf{w}^{(q-1)}, \mathbf{c}^{(q-1)})$. In case there does not exist a DP-cell $(q-1, \mathbf{z}^{(q-1)}, \mathbf{w}^{(q-1)}, \mathbf{c}^{(q-1)})$ which can be extended to the DP-cell $(q, \mathbf{z}^{(q)}, \mathbf{w}^{(q)}, \mathbf{c}^{(q)})$, the latter DP-cell is filled with a 'NO'-entry.

The DP-table is filled inductively, phase by phase, until each cell in the DP-table is filled. The idea of the proof of the lemma below is to show, for any machine i, the equivalence between the inequalities $\left\|\sum_{\tau \in \mathcal{T}_i} v'(i,\tau)\right\|_\infty \leq 1 + \epsilon$ on the one hand, and condition (4.19) on the other hand.

Lemma 4.4.6. *For phase q, there exists a DP-cell of the form $(q, \mathbf{z}^{(q)}, \mathbf{w}^{(q)}, \mathbf{c}^{(q)})$ with a 'YES'-entry if and only if there exists task assignment $\mathcal{T}$ of the first q tasks to the machines, such that for each $i \in M$ it holds that $\left\|\sum_{\tau \in \mathcal{T}_i} v'(i,\tau)\right\|_\infty \leq 1 + \epsilon$.*

Proof. I first prove the *if-part*. Consider some phase q and assume that the statement is correct for all phases $q' < q$. Let k be such that $\tau_q \in G_k$. As no first job of any task in G_k has its deadline before $(1+\epsilon)^k$ it follows by induction that $\sum_{\tau \in \mathcal{T}_i} v'(i,\tau)_\ell \leq 1 + \epsilon$ for all $\ell : 0 \leq \ell < k$. Next, I show the statement for phase q and the corresponding dimension k. Consider DP-cell of the form $(q, \mathbf{z}^{(q)}, \mathbf{w}^{(q)}, \mathbf{c}^{(q)})$ with a 'YES'-entry. I denote by $(4.19)^{(q)}$ the corresponding inequality given in (4.19) at phase q of the DP, that is when task q is being assigned to a machine. For all machines i it follows that,

$$\sum_{\tau \in \mathcal{T}_i} v'(i,\tau)_k = \sum_{k'=0}^{k-L^{(k)}} \sum_{\tau \in \mathcal{T}_i \cap G_{k'}} u'_{i,\tau} + \sum_{k'=k-L^{(k)}+1}^{k} \sum_{\tau \in \mathcal{T}_i \cap G_{k'}} v'(i,\tau)_k$$

$$= z_i^{(q)} + \sum_{k'=k-L^{(k)}+1}^{k} c_{i,k',k}^{(q)} \overset{(4.19)^{(q)}}{\leq} 1+\epsilon.$$

The last inequality follows by setting the parameter k'' of $(4.19)^{(q)}$ equal to k. Next, consider the dimension $\ell : k < \ell \leq k+L$. Note that $d_{\tau_q} < (1+\epsilon)^{k+1}$. Consequently, every task in $\cup_{k'=0}^{k} G_{k'}$ has its first deadline before $(1+\epsilon)^{k+1}$, that is, being in phase q of the DP where $\tau_q \in G_k$ it follows that $G_{k'} = \emptyset$ for all $k' > k$ for the time being. This insight is used in the first equality of the following sequence of (in)equalities.

$$\sum_{\tau\in\mathcal{T}_i} v'(i,\tau)_\ell = \sum_{k'=0}^{\ell-L^{(\ell)}} \sum_{\tau\in\mathcal{T}_i\cap G_{k'}} u'_{i,\tau} + \sum_{k'=\ell-L^{(\ell)}+1}^{k} \sum_{\tau\in\mathcal{T}_i\cap G_{k'}} v'(i,\tau)_\ell$$
$$= \left(z_i^{(q)} + \sum_{k'=k-L^{(k)}+1}^{\ell-L^{(\ell)}} w_{i,k'}^{(q)} \right) + \sum_{k'=\ell-L^{(\ell)}+1}^{k} c_{i,k',\ell}^{(q)} \overset{(4.19)^{(q)}}{\leq} 1+\epsilon.$$

The last inequality follows by setting the parameter k'' of $(4.19)^{(q)}$ equal to ℓ. Finally, the analysis for any dimension $\ell > k+L$ is equal to that of the dimension $\ell = k+L$ as the contribution of each task to any dimension $\ell \geq k+L$ will be approximated by its utilization. Thus, $\sum_{\tau\in\mathcal{T}_i} v'(i,\tau)_\ell \leq 1+\epsilon$ for all ℓ, and the *if-part* follows.

To prove the other part, consider the assignment $\mathcal{T}^{(q)}$ where the first q tasks are assigned to the machines such that $\left\| \sum_{\tau\in\mathcal{T}_i^{(q)}} v'(i,\tau) \right\|_\infty \leq 1+\epsilon$ for all machines $i \in M$. Let $\tau_q \in G_k$. Define the following values for each machine i: $z_i^{(q)} := \sum_{k'=0}^{k-L^{(k)}} \sum_{\tau\in\mathcal{T}_i^{(q)}\cap G_{k'}} u'_{i,\tau}$, $w_{i,k'}^{(q)} := \sum_{\tau\in\mathcal{T}_i^{(q)}\cap G_{k'}} u'_{i,\tau}$ for all $k' \in \{k-L^{(k)},\ldots,k\}$, and $c_{i,k',k''}^{(q)} := \sum_{\tau\in\mathcal{T}_i^{(q)}\cap G_{k'}} v'(i,\tau)_{k''}$, for all $k'' \in \{k,\ldots,k+L\}$, and all $k' \in \{k''-L^{(k'')},\ldots,k\}$. First, the DP-cell $(q,\mathbf{z}^{(q)},\mathbf{w}^{(q)},\mathbf{c}^{(q)})$ needs to exist, that is, I need to check whether the inequality $(4.19)^{(q)}$ holds. From the lemma statement it is known that $\sum_{\tau\in\mathcal{T}_i} v'(i,\tau)_\ell \leq 1+\epsilon$ for all ℓ. Thus, in particular, this inequality also holds for all dimensions $k'' \in \{k,\ldots,k+L\}$. Therefore, for each machine i and each dimension $k'' \in \{k,\ldots,k+L\}$ it holds that

$$z_i^{(q)} + \sum_{k'=k-L^{(k)}+1}^{k''-L^{(k'')}} w_{i,k'}^{(q)} + \sum_{k'=k''-L^{(k'')}+1}^{k} c_{i,k',k''}^{(q)}$$
$$= \sum_{k'=0}^{k''-L^{(k'')}} \sum_{\tau\in G_{k'}} v'(i,\tau)_{k''} + \sum_{k'=k''-L^{(k'')}+1}^{k} c_{i,k',k''}^{(q)}$$
$$= \sum_{k'=0}^{k''-L^{(k'')}} \sum_{\tau\in G_{k'}} v'(i,\tau)_{k''} + \sum_{k'=k''-L^{(k'')}+1}^{k} \sum_{\tau\in G_{k'}} v'(i,\tau)_{k''}$$
$$= \sum_{\tau\in\mathcal{T}_i^{(q)}} v'(i,\tau)_{k''} \leq 1+\epsilon.$$

Consequently, the DP-cell $(q,\mathbf{z}^{(q)},\mathbf{w}^{(q)},\mathbf{c}^{(q)})$ exists. Since the DP-cell $(q,\mathbf{z}^{(q)},\mathbf{w}^{(q)},\mathbf{c}^{(q)})$ trivially extends a DP-cell $(q-1,\mathbf{z}^{(q-1)},\mathbf{w}^{(q-1)},\mathbf{c}^{(q-1)})$ by assigning task τ_q

to machine i if $\tau_q \in \mathcal{T}_i^{(q)}$. By induction it follows that the DP-cell $(q, \mathbf{z}^{(q)}, \mathbf{w}^{(q)}, \mathbf{c}^{(q)})$ contains a 'YES'-entry, for all $q \in \{0, 1, \ldots, n\}$. □

Combining the Lemmas 4.4.1, 4.4.2, 4.4.4 and 4.4.6, and Property 4.4.3 yields a $(1 + 10\epsilon)$-approximation test, for any ϵ and a constant number of machines. The claim on the running time follows from Property 4.4.5. Redefining ϵ yields the main theorem.

Theorem 4.4.7. *For any $\epsilon > 0$ there exists a $(1 + \epsilon)$-approximation test if the number of machines is constant and which runs in time polynomial in the number of tasks.*

Proof. I show that for a constant number of unrelated machines, there exists an algorithm which either concludes that the task system is infeasible, or which returns an assignment of tasks to machines which is feasible with a speedup factor of $1 + 10\epsilon$. The running time is polynomial in n, given that m and ϵ are constants. Finally, redefining ϵ yields the desired result.

First, suppose that there is a DP-cell of the form $(n, \mathbf{z}, \mathbf{w}, \mathbf{c})$ containing a 'YES'-entry. Due to Lemma 4.4.6 there is an assignment $\mathcal{T}$ of all tasks to the machines such that $\left\|\sum_{\tau \in \mathcal{T}_i} v'(i, \tau)\right\|_\infty \leq 1 + \epsilon$, for each machine $i \in M$. By Lemma 4.4.4 this implies that $\left\|\sum_{\tau \in \mathcal{T}_i} v(i, \tau)\right\|_\infty \leq 1+2\epsilon$. Due to Property 4.4.3 this implies that $\mathrm{dbf}^*_{i,\mathcal{T}}((1+\epsilon)^k) \leq (1+2\epsilon)(1+\epsilon)^k$ for each $k \in \mathbb{N}$. It follows from Lemma 4.4.2 that $\mathrm{dbf}^*_{i,\mathcal{T}}(s) \leq (1 + 2\epsilon)^2 s$ for all $s > 0$. Finally, Lemma 4.4.1 implies that the computed task assignment is feasible if the machines run with speed $(1 + \epsilon)(1 + 2\epsilon)^2 \leq 1 + 10\epsilon$ (as $\epsilon < 1/2$).

On the other hand, if all DP-cells of the form $(n, \mathbf{z}, \mathbf{w}, \mathbf{c})$ have a 'NO'-entry, then, by Lemma 4.4.6, for any task assignment $\mathcal{T}$ there must be a machine i with $\left\|\sum_{\tau \in \mathcal{T}_i} v'(i, \tau)\right\|_\infty > 1 + \epsilon$. By Lemma 4.4.4 this yields that also $\left\|\sum_{\tau \in \mathcal{T}_i} v(i, \tau)\right\|_\infty > 1 + \epsilon$. Property 4.4.3 yields that for this machine i there exists a time moment s which is a power of $(1 + \epsilon)$ such that $\mathrm{dbf}^*_{i,\mathcal{T}}(s) > (1 + \epsilon)s$. Finally, (the negation of) Lemma 4.4.1 yields that there is a time moment r for which $\mathrm{dbf}_{i,\mathcal{T}}(r) > r$. Since, for any partition $\mathcal{T}$, there exist a machine i and a time instant r such that $\mathcal{T}$ violates the feasibility condition, it follows that the task system T is infeasible if the machines run at unit speed.

The claim that the running time is polynomial in n for given constant m and ϵ follows from Property 4.4.5 and the fact that each entry of the table can be decided upon in polynomial time. Herewith, note that $L = O(\log_{1+\epsilon}(1/\epsilon^2))$ which is constant if ϵ is constant. □

4.5 Concluding remarks

For an arbitrary number of machines, Section 4.3 describes a feasibility test where the price to pay is a speedup factor of $8 + 6\sqrt{2/3}$. On the other hand, no polynomial time feasibility test can exists which has a speedup factor strictly less than 2, unless $\mathcal{P} = \mathcal{NP}$. I believe that neither upper bound nor lower bound is the true bound.

Roughly speaking, the algorithm presented in Section 4.3 can be decoupled into three different parts: first the demand bound function is only checked at time points which are a power of 2 (or ρ) which adds a speedup factor of 2 (or ρ); next the contribution of tasks which have a small deadline with respect to the current time point are captured by their realization which adds a factor of roughly 2 as well; finally rounding the relaxation back to an integer

solution adds a factor of 3 to the approximation factor. To achieve a better upper bound I would say that the whole algorithm needs to integrated. From my point of view, there is not much more gain by tackling these three steps individually.

I believe that any feasibility test should have a speedup cost higher than two, making the problem provably harder than minimizing the makespan on unrelated parallel machines. I want to remark here that the lower bound example presented barely uses that we are delaing with tasks instead of jobs. The tasks associated with the elements in sets B and C comprise of only one job as their periods are set to infinity. Hence, these tasks do not infere later in time which is however a serious difficulty to overcome in the proof of the upper bounds.

Chapter 5

Realtime scheduling on identical machines

This chapter studies the problem of finding a partition for feasibly scheduling tasks on m *identical* parallel machines. This problem is *co-$\mathcal{NP}$-hard* even in the case of a single machine, and hence the focus has been on finding α-approximation tests. Currently, the best known result is by Chen and Chakraborty (2011) who provide a $(3 - \frac{1}{m})$-approximation test for finding a feasible partition. Some polynomial-time approximation schemes are also known in rather restricted cases, and the most general recent result in this direction is due to Chen and Chakraborty (2012) who give a polynomial-time approximation scheme for the case that the ratio of the largest to the smallest relative deadline of the tasks, λ, is bounded by a constant. However, their algorithm's running time has a super-exponential dependence on λ and hence does not extend to larger values of λ. The current chapter designs an approximation scheme with a substantially improved running time dependence on λ. In particular, the proposed algorithm only has an exponential dependence on $\log \lambda$ and hence gives a quasi-polynomial-time approximation scheme even when λ is polynomially bounded.

The final section makes a minor switch of topic and studies sporadic tasks on a single machine. To be more explicit, it shows that in the presence of smoothing, the Earliest Deadline First policy returns in expected polynomial time whether a task system is feasible on a single machine.

This chapter, up to the last section, is based on joint work with Nikhil Bansal, Suzanne van der Ster, Tjark Vredeveld and Ruben van der Zwaan.

5.1 Introduction

Problem Definition. This chapter studies the feasibility question of scheduling a sporadic task system on identical parallel machines. I refer to Subsection 1.2.2 and to the first two sections of Chapter 4 for introducing the problem of scheduling sporadic tasks on parallel machines. Like in the previous chapter, I study partitioned machine scheduling, that is, tasks are partitioned among the machines and each job should be scheduled on the machine to which its task has been assigned. The Earliest Deadline First policy (EDF) will again be the run-time scheduling algorithm of choice on each machine because EDF is optimal for

preemptive single processor scheduling and the policy is practical in implementation. I repeat that a partition $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_m\}$ of tasks to machines is *feasible* if (i) each job is be scheduled on the machine to which its corresponding task is assigned, and (ii) if EDF produces a schedule in which each job is completed before its deadline. As the problem of determining whether EDF produces a feasible schedule on a single processor is already *co-$\mathcal{NP}$-hard*, see [43], the focus in this chapter is on finding an efficient α-approximation test.

Although the realtime scheduling problem on identical parallel machines is easier than the setting of unrelated parallel machines which was studied in Chapter 4, the problem studied here is still a hard one. For example, consider the special case where the tasks' periods tend to infinity, essentially leaving a single job per task. Restrict this special case even more by setting all deadlines are equal, i. e., $d_\tau = D$ for all tasks $\tau \in T$ and some constant $D > 0$. The feasibility question is then identical to the decision version of the MAKESPAN MINIMIZATION problem on identical machines: the feasibility question looks for a partition $\mathcal{T}$ of tasks to machines such that $\sum_{\tau \in \mathcal{T}_i} c_\tau \leq D$ for all machines $i \in M$. The makespan minimization problem is known to be strongly *$\mathcal{NP}$-hard* [53] and admits a so-called polynomial-time approximation scheme (PTAS) [65]. The strong $\mathcal{NP}$-hardness implies that the existence of an FPTAS is ruled out and hence any algorithm must incur a super-polynomial dependence on the accuracy parameter ϵ, unless $\mathcal{P} = \mathcal{NP}$. When deadlines may differ among tasks, the setting is even more complex: the jobs must be distributed among the machines such that each job meets its personalized deadline. Even with a $(1+\epsilon)$ resource augmentation, this suggests that one has to look at $\log_{(1+\epsilon)} d_{\max}/d_{\min}$ deadline classes separately and ensure that jobs are scheduled such that none of these classes are overloaded on any machine. This approach already results in a running time which has an exponential dependency on $\log_{(1+\epsilon)} \lambda$, where $\lambda := d_{\max}/d_{\min}$.

Related work. Section 4.1 already covers most of the related work for scheduling sporadic tasks on parallel machines. Therefore, I focus here on the design of approximation schemes for finding a partition of the tasks on identical parallel machines. Baruah [12] provided a PTAS for partitioned scheduling on identical parallel machines for the case that the ratios of the maximum to the minimum execution time, of the maximum to the minimum period and of the maximum to the minimum deadline are all bounded by a constant. This result has recently been improved by Chen and Chakraborty [31] who provided a PTAS only requiring that the ratio of the maximum relative deadline to the minimum relative deadline, λ, is a constant. Their algorithm only checks on a limited number of time points to validate whether the schedule returned by EDF is approximately feasible by making use of an approximate demand bound function. The number of time points to be checked is in the order of $\log_{(1+\epsilon)} \lambda$ for a given constant $\epsilon > 0$. They adopt a polynomial-time approximation scheme for vector scheduling by Chekuri and Khanna [29]. For a constant ϵ, the PTAS of Chen and Chakraborty runs in $O\left(n^{(\log\log\lambda)^{\log\lambda}}\right)$ time and is thus doubly exponential in $\log\lambda$, but is still polynomial for constant λ.

For the case that the number of machines, m, is constant, Section 4.4 of the previous chapter shows a polynomial-time approximation scheme, even if the execution time of a task is machine-dependent. There, my co-authors and me approximate the demand bound function dbf by the function dbf* and use sliding windows so that the dbf only needs to be checked for a

constant, L, number of time points. In fact, each task has only $O(1/\epsilon)$ non-trivial dimensions. Using this idea of the sliding windows in the vector scheduling problem, one can (non-trivially) show that the results in [31] can be improved to a running time of $O(m^{(\log \lambda)^{O(1/\epsilon)}})$.

Contribution. This chapter proposes a $(1+\epsilon)$-feasibility test which has a lower running time than the $(1+\epsilon)$-feasibility test proposed by Chen and Chakraborty [31]. In particular, the running time of the test has a lower dependency on the ratio of deadlines, λ. To design this test, known results are efficiently combined, e. g., reducing the problem to a special case of vector scheduling [31, 29], separating the treatment of small and large tasks [31], and using a sliding window [87]. In addition, several structural properties of the periodic nature of jobs of the same task are exploited. In particular, the utilization of a task is tracked with a different accuracy than the accuracy with which the task's contribution to the demand bound function is tracked. The final contribution is a $(1+\epsilon)$-feasibility test with a running time of $O(m^{f(\epsilon)\log\lambda})$. Thus, the running time depends in the exponent only on $\log\lambda$ instead of a polynomial or even an exponential dependence on $\log\lambda$. When λ is polynomially bounded, the algorithms yields a quasi-polynomial-time approximation scheme.

The final section considers the *co-$\mathcal{NP}$-hard* problem of deciding whether a task system is feasible on a single machine. The main contribution there is that once the processing requirements of tasks have been smoothed, the expected running time of EDF is polynomial in the number of tasks and in the smoothing parameter.

Outline. The next section briefly recaptures the most important concepts and some preliminary properties in realtime machine scheduling. At the end of the section the VECTOR SCHEDULING problem is formally introduced. Section 5.3 reduces the scheduling problem to a special version of the vector scheduling problem using insights from Section 4.4 of the previous chapter and from [31]. Section 5.4 solves the special vector scheduling problem. Section 5.5 agglomerates all intermediate results and provides the final $(1+\epsilon)$-feasibility test. The final Section 5.6 questions whether EDF will result in a feasible schedule on a single machine. I show that EDF runs in expected polynomial time when tasks' processing requirements have been smoothed.

5.2 Preliminaries

In this chapter I consider identical machines and hence a task τ is characterized by the processing requirement c_τ, the relative deadline d_τ and the period p_τ. Also, the utilization of a task τ is no longer machine dependent and hence denoted by $u_\tau := c_\tau/p_\tau$. Finally, the demand bound function dbf is no longer machine depending either. Hence, I define the demand bound function $\mathrm{dbf}_\tau(s)$ of a task τ at time s as

$$\mathrm{dbf}_\tau(s) := \max\left\{0;\ \lfloor (s + p_\tau - d_\tau)/p_\tau \rfloor\, c_\tau\right\}.$$

It follows from Baruah et al. [15] that:

Property 5.2.1. *An assignment $\mathcal{T} = \{\mathcal{T}_i\}_{i \in M}$ is feasible for task system T if and only if for all $i \in M$, and for all $s > 0$*

$$dbf_{\mathcal{T}_i}(s) := \sum_{\tau \in \mathcal{T}_i : d_\tau \leq s} dbf_\tau(s) = \sum_{\tau \in \mathcal{T}_i : d_\tau \leq s} \max\left\{0; \left\lfloor \frac{s + p_\tau - d_\tau}{p_\tau} \right\rfloor c_\tau \right\} \leq s.$$

As hinted in Property 4.2.6, the above condition can be strengthened by checking only up to the hyperperiod. Denote by p_{lcm} the hyperperiod, i. e., the least common multiple, of all tasks' periods.

Property 5.2.2. *[15] A partition $\mathcal{T}$ is feasible on identical parallel machine if and only if for all machines $i \in M$ it holds that $dbf_{\mathcal{T}_i}(s) \leq s$, for all time points s such that s is the deadline of a job and $s \leq p_{lcm}$.*

The preceding property yields a exponential time feasibility test for testing whether an assignment $\mathcal{T}$ can be feasibly scheduled on a set of unit-speed machines.

In the remainder of the chapter I assume without loss of generality, conform the synchronous arrival sequence, that the jobs of a task τ are released at times $0, p_\tau, 2p_\tau, 3p_\tau, \ldots$. Further, I rescale the deadlines and also order tasks to their relative deadlines such that $1 = d_1 \leq \ldots \leq d_n$.

This chapter aims to develop a $(1+\epsilon)$-approximation scheme for any small $\epsilon > 0$. In the next section, I transform the realtime scheduling problem to the vector scheduling problem, which is defined below. Recall that $\|\boldsymbol{a}\|_\infty = \max_i \{a_i\}$ for any vector $\boldsymbol{a}$.

VECTOR SCHEDULING: Given is a set A of vectors $\boldsymbol{a} \in [0,1]^d$ and a number m. A valid solution is partition of A into m sets $A_1, ..., A_m$. The objective is to minimize the *height* of the schedule, i. e., to minimize $\max_{1 \leq i \leq m} \left\| \sum_{\boldsymbol{a} \in A_i} \boldsymbol{a} \right\|_\infty$.

5.3 Reduction to vector scheduling

The previous section describes how the demand bound function can be used to check feasibility. Property 5.2.2 requires the dbf to be checked at possibly an exponential number of points in time, namely all deadlines up to p_{lcm}. In the current section it is shown that it is approximately sufficient to check only $\log_{(1+\epsilon)}(d_n/(\epsilon d_1))$ points in time for a given $\epsilon > 0$. Afterwards, it is easy to reduce this approximate version of the realtime scheduling problem to a special case of the vector scheduling problem.

In the long run a task τ uses c_τ units of time every p_τ units of time. Unfortunately though, tasks' deadlines complicate the demand bound function. The demand bound function has sharp jumps at the (absolute) deadlines $d_\tau, p_\tau + d_\tau, 2p_\tau + d_\tau, \ldots$, but the effects of these jumps become milder as time progresses. With resource augmentation, which gives $\epsilon \cdot s$ extra processing time up to time s, these sharp jumps can be ignored after a certain point in time s^*. Beyond time s^*, it is sufficient to use the *utilization* (the average processing requirement) of a task.

I reduce the scheduling problem to the vector scheduling problem by modeling each task τ as a vector w^τ. The coordinates of the vector then capture the normalized demand bound function per time point which is being check. The last dimension of each vector w^τ maps the

utilization of task τ. The feasibility question then reduces to whether the vectors w^τ can be partitioned into sets $W_1, \ldots, W_m$ such that $\sum_{w^\tau \in W_i} w_k^\tau \leq 1$ for all coordinates k and all machines $i \in M$. As w^τ denotes the normalized demand bound function, this condition implies that the demand bound function per machine i at some time point s is no more than s. The reduction to the vector scheduling problem allows for the original problem to be (approximately) solved efficiently.

Before moving to the details, I give a high-level overview of the transformation from the scheduling of sporadic tasks to vector scheduling. Some ideas of this transformation have already been applied in the previous chapter.

1. Approximate the demand bound function dbf_τ for each task τ by a modified demand bound function dbf_τ^* which is linear *after* some few initial jumps (Lemma 4.4.1).
2. Only check the modified function dbf_τ^* at time points which are powers of $(1+\epsilon)$ (Lemma 4.4.2).
3. Disregard checking dbf_τ^* before the first deadline d_1 (Property 5.3.1).
4. Encode the modified function dbf_τ^* of a task τ into vectors v^τ (Equation 5.2).
5. Remove coordinates of the vector v^τ corresponding to time points much later than the last deadline, as these are all equal to the utilization, creating the vector w^τ (Lemma 5.3.3).

The first step is to use an approximate demand bound function $\text{dbf}_\tau^*(s)$ which has been introduced in the previous chapter, see equation (4.14). The constant L was defined to be the minimum integer which satisfies $1 \leq (1+\epsilon)^{L-1}\epsilon^2$. Then, $\text{dbf}_\tau^*(t)$ is defined as:

$$\text{dbf}_\tau^*(s) = \begin{cases} \left\lfloor \frac{s+p_\tau-d_\tau}{p_\tau} \right\rfloor c_\tau & \text{if } s < (1+\epsilon)^L \cdot d_\tau, \\ u_\tau \cdot s & \text{otherwise.} \end{cases} \tag{5.1}$$

Similarly, I define $\text{dbf}_{\mathcal{T}_i}^*(s) = \sum_{\tau \in \mathcal{T}_i} \text{dbf}_\tau^*(s)$ for all time points $s > 0$ and all machines $i \in M$. The following lemma was already stated in the previous chapter and hence the proof is omitted here.

Lemma 4.4.1. *Given an assignment $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_m\}$ and let $0 < \epsilon < 1/2$. Then, for all machines $i \in M$,*

(i) if $\text{dbf}_{\mathcal{T}_i}^*(r) \leq \alpha \cdot r$ for all $r \geq 0$, then $\text{dbf}_{\mathcal{T}_i}(s) \leq (1+\epsilon) \cdot \alpha \cdot s$ for all $s \geq 0$;

(ii) if $\text{dbf}_{\mathcal{T}_i}(r) \leq \alpha \cdot r$ for all $r \geq 0$, then $\text{dbf}_{\mathcal{T}_i}^*(s) \leq (1+\epsilon) \cdot \alpha \cdot s$ for all $s \geq 0$.

The next lemma shows that the modified demand bound function dbf_τ^* only needs to be checked at time points which are a factor $(1+\epsilon)$ apart and was already shown in Section 4.4.

Lemma 4.4.2. *Consider a task τ and a machine $i \in M$. If for all $k \in \mathbb{N}_{\geq 0}$ it holds that $\text{dbf}_\tau^*((1+\epsilon)^k) \leq \alpha \cdot (1+\epsilon)^k$ then $\text{dbf}_\tau^*(s) \leq \alpha \cdot s \cdot (1+2\epsilon)$ for all $s \geq 0$.*

Observing how the functions dbf and dbf^* are defined, one more obvious property follows from rounding down the quantity $(s + p_\tau - d_\tau)/p_\tau$. This observation allows starting the feasibility analysis at the first deadline only.

Property 5.3.1. *For all tasks τ, $dbf_\tau(s) = dbf^*_\tau(s) = 0$ for all $s < d_\tau$. In particular, $dbf_\tau(s) = dbf^*_\tau(s) = 0$ for all $s < d_1$ and all tasks $\tau \in T$.*

Using Lemmas 4.4.1 and 4.4.2, and Property 5.3.1, I encode the approximate demand bound function dbf^*_τ into a vector v^τ. Therefore, only points in time that are powers of $(1+\epsilon)$ need to be checked and any time points before d_1 might be disregarded. This vector covers the timespan between the first deadline d_1 and some end point s_{end}. Property 5.2.2 would implies that $s_{end} \geq p_{lcm}$ needs to hold. Additionally, I require that $s_{end} \geq (1+\epsilon)^L d_n$. Secondly, it is important to note that I use the *normalized* demand bound function which is $\text{dbf}^*_\tau(s)/s$. Combining all aspects together enables the reduction to the Vector Scheduling problem.

For each task τ define the vector v^τ,

$$v^\tau_k := \frac{\text{dbf}^*_\tau\left((1+\epsilon)^{k-1}\right)}{(1+\epsilon)^{k-1}} \text{ for all } k \in \left[\left\lceil \log_{(1+\epsilon)}(s_{end}) \right\rceil\right]. \tag{5.2}$$

where $[n]$ represents the set of integers $\{1, \ldots, n\}$.

The following lemma connects the vector scheduling problem formally to the sporadic task system scheduling.

Lemma 5.3.2. *Define the vectors v^τ as in* (5.2)*. Given is a partition of vectors v^τ into m sets $V_1, \ldots, V_m$ and the corresponding partition of tasks $\tau \in T$ into m sets $\mathcal{T}_1, \ldots, \mathcal{T}_m$. Then, for all machines i,*

(i) if $\left\|\sum_{v^\tau \in V_i} v^\tau\right\|_\infty \leq \alpha$, then $dbf_{\mathcal{T}_i}(s) \leq (1+\epsilon)(1+2\epsilon) \cdot \alpha \cdot s$ for all $s \geq 0$;

(ii) if $dbf_{\mathcal{T}_i}(s) \leq \alpha \cdot s$ for all $s \geq 0$, then $\left\|\sum_{v^\tau \in V_i} v^\tau\right\|_\infty \leq \alpha \cdot (1+\epsilon)$.

The proof of the above lemma follows straightforwardly from combining the Lemmas 4.4.1 and 4.4.2, and the definition of v^τ_k, see (5.2). The final structural simplification is to reduce the dimension of the vectors by observing that, by definition of dbf^*_τ, for every vector each coordinate corresponding to a time point $s > (1+\epsilon)^L d_n$ is equal to its utilization. Therefore, for all vectors I may remove all dimensions $i > K$, where K is equal to $\left\lceil \log_{(1+\epsilon)} d_n/\epsilon \right\rceil + 1$. recall I rescaled the deadlines such that $d_1 = 1$. This idea to reduce the number of time points that need to be checked is due to Chen and Chakraborty [31].

These trimmed vectors will be denoted as w^τ. Let K be equal to $\left\lceil \log_{(1+\epsilon)} d_n/\epsilon \right\rceil + 1$. Then, w^τ is defined as

$$w^\tau_k := \begin{cases} v^\tau_k & \text{if } k = 1, \ldots, K-1, \\ u_\tau & \text{if } k = K. \end{cases}$$

Lemma 5.3.3 (Like Chen and Chakraborty [31])**.** *Given a partition of vectors v^τ into m sets $V_1, \ldots, V_m$ and the corresponding partition of vectors w^τ into m sets $W_1, \ldots, W_m$, and $\epsilon > 0$. Then, for all machines i,*

(i) if $\sum_{w^\tau \in W_i} w^\tau_k \leq \alpha$ for all $k \in [K]$, then $\sum_{v^\tau \in V_i} v^\tau_k \leq (1+2\epsilon)\alpha$ for all $k \in \left[\left\lceil \log_{(1+\epsilon)}(s_{end}) \right\rceil\right]$;

(ii) if $\sum_{v^\tau \in V_i} v^\tau_k \leq \alpha$ for all $k \in \left[\left\lceil \log_{(1+\epsilon)}(s_{end}) \right\rceil\right]$, then $\sum_{w^\tau \in W_i} w^\tau_k \leq \alpha$ for all $k \in [K]$.

Proof. First show property (i). As the first $K-1$ coordinates of any $v^\tau \in V$ equal the first $K-1$ coordinates of the corresponding vector w^τ, it follows that $\sum_{v^\tau \in V_i} v_k^\tau = \sum_{w^\tau \in W_i} w_k^\tau \leq \alpha$ for all $k \in [K-1]$. Next consider any $k \geq K = \lceil \log_{1+\epsilon} d_n/\epsilon \rceil + 1$. Define $s_k = (1+\epsilon)^{k-1} \geq (1+\epsilon)^{K-1} \geq d_n/\epsilon$. Then, for a given task $\tau \in T$,

$$v_k^\tau = \frac{\mathrm{dbf}_\tau^*(s_k)}{s_k} \leq \frac{u_\tau \cdot s_k + c_\tau}{s_k} = u_\tau + \frac{c_\tau}{s_k} \leq u_\tau + \epsilon \frac{c_\tau}{d_n}.$$

In the above, the first inequality follows since either $\mathrm{dbf}_\tau^*(s) = u_\tau \cdot s$ or

$$\mathrm{dbf}_\tau^*(s) = \left\lfloor \frac{s + p_\tau - d_\tau}{p_\tau} \right\rfloor c_\tau \leq \frac{s+p_\tau}{p_\tau} c_\tau = u_\tau \cdot s + c_\tau.$$

Next, define for any task $\tau \in T$ the rounded up deadline $d_\tau^+ = (1+\epsilon)^{\lceil \log_{1+\epsilon} d_\tau \rceil}$. Also note that $\mathrm{dbf}_\tau^*(s) \geq c_\tau$ for any $s \geq d_\tau$. Then, it follows that

$$v_k^\tau \leq u_\tau + \epsilon \frac{c_\tau}{d_n} \leq u_\tau + \epsilon \frac{\mathrm{dbf}_\tau^*(d_n^+)}{d_n} \leq u_\tau + \epsilon(1+\epsilon) \frac{\mathrm{dbf}_\tau^*(d_n^+)}{d_n^+}.$$

Finally, using that $\sum_{w^\tau \in W_i} w_k^\tau \leq \alpha$ for all $k \in [K]$, it holds that

$$\sum_{v^\tau \in V_i} v_k^\tau \leq \left(\sum_{v^\tau \in V_i} u_\tau + (\epsilon + \epsilon^2) \frac{\mathrm{dbf}_\tau^*(d_n^+)}{d_n^+} \right) = \sum_{v^\tau \in V_i} \left(w_K^\tau + (\epsilon+\epsilon^2) w^\tau_{\log_{1+\epsilon} d_n^+} \right) \leq \alpha + 2\epsilon\alpha.$$

Note that the above equality is valid since $\log_{1+\epsilon} d_n^+ < K-1$ as $L > 1$.

The second property (ii) follows easily. The first $K-1$ coordinates of any vector $w^\tau \in W$ equal the first $K-1$ coordinates of the corresponding vector v^τ. As I require that $s_{end} \geq (1+\epsilon)^L d_n$ it follows that the last position of any vector $v^\tau \in V$ equals u_τ, see (5.1). Therefore, the last position in any vector w^τ equals the last position in the corresponding vector v^τ. □

Now the main theorem of this section can be proven as a result of combining Lemmas 5.3.2 and 5.3.3. Here, the function $\mathrm{dbf}_{\mathcal{T}_i}(s)$ in the first property of Lemma 5.3.2 is upper bounded by $(1+2\epsilon)^2$.

Theorem 5.3.4. *Given a partition of vectors w^τ into m sets $W_1, \ldots, W_m$ and the corresponding assignment of tasks τ into m sets $\mathcal{T}_1, \ldots, \mathcal{T}_m$, and $\epsilon > 0$. Then, for all machines $i \in M$,*

(i) if $\left\| \sum_{w^\tau \in W_i} w^\tau \right\|_\infty \leq \alpha$, *then* $\mathrm{dbf}_{\mathcal{T}_i}(s) \leq (1+2\epsilon)^3 \cdot \alpha \cdot s$ *for all* $s \geq 0$;

(ii) if $\mathrm{dbf}_{\mathcal{T}_i}(s) \leq \alpha \cdot s$ *for all* $s \geq 0$, *then* $\left\| \sum_{w^\tau \in W_i} w^\tau \right\|_\infty \leq \alpha \cdot (1+\epsilon)$.

The previous theorem, tells that if one can partition the set of tasks T into sets $\mathcal{T}_1, \ldots, \mathcal{T}_m$ (or equivalently the set of vectors $w^\tau \in W$ into sets $W_1, \ldots, W_m$) such that if $\sum_{w^\tau \in W_i} w_k^\tau \leq 1+\epsilon$, for all $i \in M$ and $k \in [K]$, then one can feasibly schedule the tasks in set $\mathcal{T}_i$ on machine i if this machine receives a speedup factor $(1+2\epsilon)^4$. Therefore, the goal of the next section is to develop a procedure which schedules the vectors w^τ to the machines such that $\sum_{w^\tau \in W_i} w_k^\tau \leq 1+\epsilon$ for all machines $i \in M$ and all coordinates $k \in [K]$ in appropriate time.

5.4 Solving the special vector scheduling problem

In the previous section I formulated the realtime scheduling problem as a special case vector scheduling problem. This section develops an approximation test for this special version of the vector scheduling, specifically exploiting the properties of the vectors that are obtained in the previous section. I combine several techniques from BIN PACKING and VECTOR SCHEDULING and design a 'sliding window' dynamic programming approach (like in Section 4.4) which exploits the special nature of the vectors under consideration. Before continuing with the algorithm description, I first introduce some necessary concepts and notation.

In light of Section 5.3, the challenge is to find a partition of the tasks $\tau \in T$ into sets $\mathcal{T}_1, \dots \mathcal{T}_m$ such that $\left\|\sum_{\tau\in\mathcal{T}_i} w_k^\tau\right\|_\infty \leq 1+\epsilon$, for all machines $i \in M$, or alternatively that $\sum_{\tau\in\mathcal{T}_i} w_k^\tau \leq 1+\epsilon$, for all machines $i \in M$ and all coordinates $k \in [K]$. The main theorem of this section is the following. Its proof is postponed to conclude the current section.

Theorem 5.4.1. *Given a fixed $\epsilon > 0$ and a set of vectors W from $[0,1]^K$ as defined in Section 5.3. Then, Algorithm 2 determines in $O(m^{O(C)})$ time whether the set of vectors W can be feasibly partitioned by exceeding the load in every coordinate by at most a factor of $(1+\epsilon)$, or whether no feasible schedule exists, where $C = (\lceil(8L+19)/\epsilon\rceil)^L \cdot \lceil(8LK+19K)/\epsilon\rceil$.*

This section is structured as follows. First, I repeat the properties of the vectors under consideration and state the necessary notation and definitions for this section. In Subsection 5.4.2 a high-level overview of the algorithm is given. The different modules of the algorithm are then presented in the Subsections 5.4.3 up to 5.4.6.

5.4.1 Notation and definitions.

A task τ is associated to a vector w^τ from $[0,1]^K$ with $K := \lceil \log_{(1+\epsilon)} d_n/\epsilon\rceil + 1$. Define $k_\tau = \lceil \log_{(1+\epsilon)} d_\tau\rceil$. The vector w^τ was in the previous subsection constructed such that:

$$w_k^\tau := \begin{cases} 0 & \text{if } k \leq k_\tau - 1, \\ \frac{dbf_\tau(s_k)}{s_k} & \text{if } k = k_\tau, \dots, k_\tau + L - 1, \\ u_\tau & \text{otherwise,} \end{cases} \tag{5.3}$$

where $s_k = (1+\epsilon)^k$. A vector w^τ is of size K but actually only contains at most $L+2$ different entries. To ensure that the running time of the algorithm does not explode, I summerize the vector w^τ by only these $L+1$ positive (and possibly distinct) values. I introduce a so-called *t-configuration*.

Let $0 < \eta < 1$ be a small constant. A t-configuration is an $(L+1)$-tuple $(f_1, \dots, f_L, f_u)$ where $f_k \in \{0, \eta, 2\eta, \dots, 1-\eta, 1\}$, for all $k \in [L]$ and $f_u \in \{0, \delta, 2\delta, \dots, 1\}$ where $\delta := \eta/K$. A machine i *conforms* to a t-configuration $f = (f_1, \dots, f_L, f_u)$ if the contribution to coordinate $t-1+k$ is at most f_k, for all $k \in [L]$, and if the contribution to all coordinates $k \geq t+L$ is at most f_u. A *t-profile* is a tuple $Q = (q_1, \dots, q_m)$ where q_i is the t-configuration corresponding to machine i.

As the first L elements in a t-configuration can attain one of $\lceil 1/\eta\rceil$ different values and the last element can attain one of $\lceil 1/\delta\rceil$ different values, the following property follows:

Property 5.4.2. *The number of different t-configurations, denoted by C, is*

$$C := \left(\left\lceil \frac{1}{\eta} \right\rceil\right)^{L} \cdot \left\lceil \frac{1}{\delta} \right\rceil .$$

In part of the analysis, a t-profile Q has a alternative representation given by the tuple $\langle n_1, \ldots, n_C \rangle$ where n_f denotes the number of machines that conform to configuration f. As the n_f sum up to m, the number of different t-profiles is at most m^C.

Definition 5.4.3. *Given a t-profile $Q = (q_1, \ldots, q_m,)$ and a vector $e = (e_1, \ldots, e_L, e_u)$, I define $Q + e$ and its result $Q' = (q'_1, \ldots, q'_m)$ as the pointwise addition of the vector e to each configuration $q_i \in Q$, i. e., $q'_i = e + q_i$ for all $i \in [m]$.*

Finally, I categorize vectors based on the position of the first non-zero valued coordinate.

Definition 5.4.4. *A vector w^τ is a* t-vector *if its first non-zero coordinate is coordinate t.*

5.4.2 Overview of the algorithm.

The Vector Scheduling Algorithm (VSA), given in Algorithm 2, first applies two rounding steps as described in Section 5.4.3. After the rounding steps, two subroutines are used to see whether the tasks can be feasibly scheduled such that the load of on every coordinate for each machine is at most $1 + \epsilon$. The dynamic program *sliding window DP*, formulated in Subsection 5.4.4, determines whether there exists a schedule for all k-vectors with $k \leq t$ conforming to a given t-profile Q. This dynamic program, called $T[t, Q]$, makes use of a second algorithm $B[t, R]$, which is described in Subsection 5.4.5, that determines, for some given $t \in [K]$ whether all t-vectors can be scheduled conforming to a given t-profile. This second algorithm is almost identical to the work of Chekuri and Khanna for Vector Scheduling [29], however here I have to be careful with the analysis to show that their approach also works when different coordinates can be approximated differently.

Both algorithms need to be able to determine whether a t-profile R and a $(t-1)$-profile (or t-profile) S can be combined into a t-profile Q. The way this is done is presented in Section 5.4.6.

Algorithm 2 Vector Scheduling Algorithm (VSA)

Require: Input: a set W of vectors w^τ as defined in Section 5.3, and $\eta > 0$.

1: Define $\delta := \eta / K$.
2: For each vector w^τ round each component w^τ_k down to the nearest power of $\frac{1}{1+\eta}$.
3: Modify each vector (Lemma 5.4.5)

$$z^\tau_k := \begin{cases} 0 & \text{if } w^\tau_k \leq \delta \left\| w^\tau \right\|_\infty , \\ w^\tau_k & \text{otherwise.} \end{cases}$$

4: Compute $B[t, R]$ for all possible t-profiles R and all coordinates t, see Subection 5.4.5
5: Compute the value $T[t, Q]$ with dynamic programming, see Subsection 5.4.4.
6: Return the value $T[K, 1]$.

5.4.3 Preprocessing

In the preprocessing part, steps 2 and 3 of VSA, the vectors are rounded. First, every element of each vector is rounded down to the nearest power of $1/(1+\eta)$. The second rounding step ensures that the positive values in one vector are not more than a factor $1/\delta$ apart. For all $\tau \in T$ and all $k \in [K]$

$$z_k^\tau := \begin{cases} 0 & \text{if } w_k^\tau \leq \delta \left\| w^\tau \right\|_\infty, \\ w_k^\tau & \text{otherwise.} \end{cases} \tag{5.4}$$

The same rounding step is applied by Chekuri and Khanna [29]. They show that the vectors z^τ provide a good approximation of the vectors w^τ. The following lemma follows directly from the proof of Lemma 2.1 in [29].

Lemma 5.4.5 (Chekuri and Khanna [29])**.** *Given a set W of vectors from $[0,1]^K$, let Z be a modified set of W where each vector w^τ in W is replaced with a vector z^τ according to* (5.4)*. Then, for any subset of vectors $Z' \subseteq Z$ with corresponding subset $W' \subseteq W$, it holds that $\sum_{w^\tau \in W'} w_k^\tau \leq \sum_{z^\tau \in Z'} z_k^\tau + 3\eta$.*

Proof. Remember that $\left\|z^\tau\right\|_\infty := \max_k z_k^\tau$ and, as $z_k^\tau \geq 0$, $\left\|z^\tau\right\|_1 := \sum_k z_k^\tau$. It follows from the transformation described above that

$$\begin{aligned} \sum_{w^\tau \in W'} w_k^\tau &\leq \sum_{z^\tau \in Z'} z_k^\tau + \delta \sum_{z^\tau \in Z'} \left\|z^\tau\right\|_\infty \leq \sum_{z^\tau \in Z'} z_k^\tau + \delta \sum_{z^\tau \in Z'} \left\|z^\tau\right\|_1 \\ &= \sum_{z^\tau \in Z'} z_k^\tau + \delta \left\| \sum_{z^\tau \in Z'} z^\tau \right\|_1 \leq \sum_{z^\tau \in Z'} z_k^\tau + \delta \cdot (K+1) \cdot \left\| \sum_{z^\tau \in Z'} z^\tau \right\|_\infty \\ &\leq \sum_{z^\tau \in Z'} z_k^\tau + \delta \cdot (K+1) \cdot (1+\eta) \leq \sum_{z^\tau \in Z'} z_k^\tau + 3\eta, \end{aligned}$$

where the last inequality is true when $K \geq 2$ and $\eta \leq 1$. □

5.4.4 The sliding window dynamic program

I introduce a dynamic programming table T where a particular DP-cell is denoted by $T[t, Q]$. Here, Q denotes a t-profile. This *sliding window dynamic program* works in $K - L$ phases as it moves from the first coordinate to coordinate $K - L$. While scheduling all t-vectors in a certain phase t, the DP also looks ahead to the next $L - 1$ coordinates and the last utilization coordinate to ensure no conflicts arise in these coordinates. Hence, the DP iteratively slides a window of length L, i. e., covering L coordinates, from coordinate 1 to coordinate $K - L$ in as many phases.

Intuitively, phase t corresponds to scheduling the t-vectors to a partial schedule for all k-vectors with $k < t$. The dynamic programming cell $T[t, Q]$ evaluates to 'TRUE' if all k-vectors for $k \leq t$ can be assigned to machines such that the machines conform to the t-profile Q, and evaluates to 'FALSE' otherwise. To determine the value of DP-cell $T[t, Q]$, the t-profile Q is *split* into a t-profile R and a $(t-1)$-profile S which respectively capture the division of space per machine and per coordinate between on the one hand the t-vectors, and the other hand

k-vectors with $k < t$. How t-profiles can be split will be explained in detail in Subsection 5.4.6. For now it is sufficient to know that the splitting of a t-profile can be done in time $O(m^{O(C)})$, see Lemma 5.4.10.

Since the t-configurations are 'coarse valued' (all values are either multiples of η or δ), it is unclear how to split the t-profile Q: perhaps a coordinate f_k of the t-configuration can be split into two parts yielding a feasible t-profile R and a $(t-1)$-profile S but not in such a way that the two parts are multiples of η. In that case, the corresponding DP-cell is erroneously evaluated to false. To circumvent this issue, an additional small error in each phase of the sliding window DP is allowed. Then, if a coordinate f_k would need to be split in non-multiples of η, the DP can now do so and then round both counterparts up to a multiple of η. For this reason the vector $(\boldsymbol{\eta}, \delta) = (\eta, \ldots, \eta, \delta)$ is added to the t-profile S.

To determine the value for $T[t, Q]$, one needs to know whether or not all t-vectors can be scheduled conform to profile R. For now, assume that for all t and R, the function $B[t, R]$ determines in constant time whether this is possible or not. In Subsection 5.4.5 I describe how these values can be pre-computed so that the DP can indeed access them in constant time for the computations of $T[t, Q]$.

Given the procedure $B[t, R]$, the recursive formula for T can be easily computed by considering all possible combinations of t-profiles R and $(t-1)$-profiles S that (i) can be combined to a t-profile Q; that (ii) determine whether or not all t-vectors can be scheduled conforming R and that (iii) can schedule all other k-vectors with $k < t$ conforming S. That is, for $t \in \mathbb{N}_{\geq 0}$,

$$T[t, Q] = \bigvee_{(R,S) \in \mathcal{W}(Q)} (B[t, R] \wedge T[t-1, S + (\eta, \ldots, \eta, \delta)]), \tag{5.5}$$

where $\mathcal{W}(Q)$ contains all tuples (R, S) of t-profiles R and $(t-1)$-profiles S that Q can be split into, and approximation test $B[t, R]$ evaluates 'TRUE' if all t-vectors *can be* scheduled conforming to t-profile $R + (2\eta, \ldots, 2\eta, (L+2)\delta)$, and evaluates 'FALSE' if the t-vectors *cannot* be scheduled conforming to t-profile R. Recall that the vector $(\boldsymbol{\eta}, \delta)$ has been added to the $(t-1)$-profile S to allow for easier splitting of Q. The base case of the recursion is

$$T[0, Q] = B[0, Q]. \tag{5.6}$$

To evaluate the running time of computing $T[K, Q]$, I remark once again that the subprocedure $B[t, R]$ is precomputed and can be accessed in $O(1)$ time, see Subsection 5.4.5.

Lemma 5.4.6. *Let W be a set of vectors w^τ as defined in* (5.3) *and let $\eta > 0$ be small enough. Algorithm 2 decides in $O(K \cdot m^{O(C)})$ time whether there exists a partition of the vectors W into m sets $W_1, \ldots, W_m$ such that $\left\|\sum_{w^\tau \in W_i} w^\tau\right\|_\infty < 1 + (8L+19)\eta$ for all $i \in [m]$, or that there exist no partition $W_1, \ldots, W_m$ for which $\left\|\sum_{w^\tau \in W_i} w^\tau\right\|_\infty \leq 1$ for all $i \in [m]$.*

Proof. First I focus on the running time of the procedure which is dominated by steps 4 and 5 of VSA. In Lemma 5.4.9 of Subsection 5.4.5 the running time of step 4 is proven to be $O(m^{O(C)})$. Note that there exist C different t-configurations (Property 5.4.2). As there are at most m machines having a certain t-configuration, there are at most m^C different t-profiles and therefore the dynamic program has $O(Km^C)$ different states. For evaluating a recursive step, each possible split of a t-profile into a new t-profile and a $(t-1)$-profile is considered.

Since there exist at most $m^C m^C$ such combinations and splitting takes $O(m^{O(C)})$ time (Lemma 5.4.10, Subsection 5.4.6), the total running time is $O(K \cdot m^{O(C)})$.

To show correctness, note that in each recursive step an error is introduced due to the addition of a vector $(\boldsymbol{\eta}, \delta)$. Furthermore, the computation of $B[t, R]$ also incurs an error, as it only decides whether all t-vectors can be scheduled conforming to $R+(2\eta, \ldots, 2\eta, (L+2)\delta)$. Fix any coordinate c on any machine and consider the total error on this coordinate. In each phase t for $c - L \leq t \leq c$ an additional error of $\eta + 2\eta$ is introduced and in each phase $t \leq c - L - 1$ an additional error of $\delta + (L + 2)\delta$ by the 'utilization' of the vectors assigned in those phases. Therefore, the maximum error is at most $(L+1)3\eta + K(L+3)\delta \leq (4L+6)\eta$ for each coordinate on any machine, as $\delta = \eta/K$.

Finally, consider the error that the rounding procedures might introduce. The first rounding procedure rounds each entry of the vector w^τ down to a power of $\frac{1}{1+\eta}$. The resulting error per coordinate per machine is at most a factor $1 + \eta$. The second modifying step rounds w_k^τ down to zero in case $w_k^\tau \leq \delta \left\| w_\tau \right\|_\infty$. Lemma 5.4.5 shows that for this second rounding step the corresponding error for the final schedule amounts to an additional factor of at most 3η per coordinate per machine. The total error incurred by the final step 5 amounts to at most $(4L+6)\eta$. Thus, the accumulated error is bounded by a factor $(1 + \eta)(1 + (4L + 9)\eta) \leq 1 + (8L + 19)\eta$ for $\eta \leq 1$. □

Lemma 5.4.6 now enables to prove the main Theorem of this section.

Proof of Theorem 5.4.1. The theorem follows easily from Lemma 5.4.6 above. Setting $\eta := \epsilon/(8L + 19)$ leads to partition with height at most $1 + (8L + 19)\eta = 1 + \epsilon$. The running time is dominated by Algorithm 2. By plugging in all parameters K, L, C, η and δ a running time of $O\left(K \cdot m^{O(g(\epsilon)\cdot\log(\lambda))}\right)$, is attained where $g(\epsilon)$ and K are functions depending on ϵ only.

$$
\begin{aligned}
O(K \cdot m^{O(z)}) &= O\left(K \cdot m^{O\left(\left(\left\lceil \frac{1}{\eta} \right\rceil\right)^L \cdot \left\lceil \frac{1}{\delta} \right\rceil\right)}\right) = O\left(K \cdot m^{O\left(\left(\frac{1}{(\epsilon/(8L+19))}\right)^L \cdot \frac{K}{(\epsilon/(8L+19))}\right)}\right) \\
&= O\left(K \cdot m^{O\left(\left(\frac{L}{\epsilon}\right)^L \cdot \frac{L\cdot K}{\epsilon}\right)}\right) = O\left(K \cdot m^{O\left(\left(\frac{\log_{1+\epsilon}\frac{1}{\epsilon}}{\epsilon}\right)^{\log_{1+\epsilon}\frac{1}{\epsilon}} \cdot \frac{\log_{1+\epsilon}\frac{1}{\epsilon}\cdot\log_{1+\epsilon}\frac{\lambda}{\epsilon}}{\epsilon}\right)}\right) \\
&=: O\left(K \cdot m^{O(f(\epsilon)\cdot\log(\lambda))}\right).
\end{aligned}
$$

□

5.4.5 A subprocedure for scheduling t-vectors

In previous subsection the special case vector scheduling problem is solved assuming that the algorithm VSA can call the subprocedure $B[t, R]$ in constant time, i. e., assuming that there exists a table B telling whether all t-vectors can be scheduled satisfying some t-profile R. Denote the set of all t-vectors z^τ in Z by Z_t. In this subsection, I describe the function $B[t, R]$ which determines either whether (i) all vectors in Z_t can be scheduled conforming to t-profile $R + (2\eta, ..., 2\eta, (L + 2)\delta)$, or whether (ii) they cannot be scheduled conforming to R. The pseudo-code for $B[t, R]$ is given in Algorithm 3 and will be shown to run in $O\left(m^{O(\frac{1}{\eta\delta})}\right)$

time. The function $B[t, R]$ is calculated prior to the execution of the DP-table T, see step 4 of Algorithm 2, and hence can be consulted in constant time by T.

In phase t of the sliding window DP, the DP is only concerned about coordinates t until $t + L - 1$ and the last coordinate which captures the utilization. Therefore, in the current subsection, I discard all remaining coordinates of the vector z^τ for all $z^\tau \in Z_t$, i. e., I act as if z^τ only has dimension $L + 1$ (namely those corresponding to the coordinates t until $t + L - 1$ and the final coordinate $K+1$). The idea of Algorithm 3 is the following: to partition the set Z_t into a set of small vectors Z_t^{small} and a set of big vectors Z_t^{big}. Further, split the t-profile R into two t-profiles R^{big} and R^{small}. If all vectors in Z_t^{big} can be scheduled conform t-profile R^{big} and if all vectors in Z_t^{small} can be scheduled conform t-profile R^{small} then $B[t, R]$ returns 'TRUE'. In no such combinations can be found, the procedure should return 'FALSE'. As t-profiles are only tracked with accuracy of η, I add the vector $(\boldsymbol{\eta}, \delta)$ to the t-profile R^{small} when determining whether the vectors in Z_t^{small} can be scheduled (conform (5.5) in Section 5.4.4). As the vectors in R^{big} are big, the number of these vectors is bounded and they can be scheduled using a dynamic program. I will call this DP the *inner DP*. The small vectors will be scheduled using an LP-relaxation and the appropriate rounding thereof.

Algorithm 3 Scheduling the vectors in Z_t to a t-profile R

Require: Given profile R, vectors Z_t from $[0, 1]^{L+1}$, $\eta \in [0, 1]$ and $\delta := \eta/K$.
1: Let $Z_t^{\text{big}} = \{z^\tau \in Z_t \,:\, \|z^\tau\|_\infty > \delta\}$ and $Z_t^{\text{small}} = Z_t \setminus Z_t^{\text{big}}$.
2: **for all** t-profiles R^{small} and R^{big} that R can be split into **do**
3: Return `true` if and only if the following two statements are both true:
4: (i) Z_t^{big} can be scheduled conforming to $R^{\text{big}} + (\eta, \ldots, \eta, \delta)$, see Lemma 5.4.7;
5: (ii) Z_t^{small} can be scheduled conforming to $R^{\text{small}} + (\eta, \ldots, \eta, (L + 1)\delta$, see Lemma 5.4.8.
6: **end for**
7: Return `false`.

A vector $z^\tau \in Z_t$ is called *big* if $\|z^\tau\|_\infty > \delta$, and *small* otherwise. Lemma 5.4.5 then yields that all coordinates of a big vector z^τ are at least δ^2. I deduce that the number of vectors in Z_t^{big} is bounded. Therefore, the number of DP cells is bounded as well. There are as many phases in the inner DP as there are machines. Phase i of the inner DP considers the feasibility question whether all vectors in a subsets of Z_t^{big} can be scheduled conform t-profile R^{big} while using only the first i machines.

Lemma 5.4.7 (Chekuri and Khanna [29])**.** *Given $(L + 1)$-dimensional* big *vectors Z_t^{big} and a t-profile R^{big}. There is an algorithm that determines in $O\left(m^{O\left(\frac{1}{\delta\eta}\right)}\right)$ time whether there exists a schedule that conforms to t-profile R^{big} or outputs that there is no schedule conforming to R^{big}.*

Proof. As big vectors are considered, the smallest non-zero coordinate of each vector is at least δ^2, see Lemma 5.4.5. Further, by step 2 of Algorithm 2, each coordinate is rounded down to a power of $1/(1 + \eta)$. Therefore, each coordinate $1 \leq k \leq L + 1$ can take at most $\log_{(1+\eta)}(1/\delta^2)$ different values. It follows that there are at most $(\log_{(1+\eta)}(1/\delta^2))^{L+1}$ different types of vectors. Notice that at most L/δ big vectors vectors can be scheduled on each machine. Hence, there

are at most mL/δ big vectors as otherwise it can be safely outputted that there is no schedule conforming to any profile.

Define the recurrence $DP[i, V]$ such that $DP[i, V]$ is true if the vectors $V \subseteq Z_t^{\text{big}}$ can be scheduled on machines 1 to i conform to t-profile R^{big}, that is, conform to the corresponding t-configurations $r_1, \ldots, r_i$:

$$DP[i, V] = \bigcup_{V' \subseteq V : V' \text{ on machine } i \text{ according to } r_i} DP[i-1, V \setminus V'].$$

Above the union is taken over all sets $V' \subseteq V$ such that the vectors z^τ in V' can be scheduled on machine i conform the t-configuration r_i of machine i. This condition can easily be checked by validating that $\sum_{z^\tau \in V'} z_k^\tau \leq r_{i,k}$ for all $k \in \{1, \ldots, L+1\}$. (Recall that I am abusing notation here as the I assumed in this subsection that the vector z^τ has only dimension $L+1$ and hence z_k^τ can directly be compared with $r_{i,k}$.) The base case $DP[1, V]$, which asks whether the vectors in V can be scheduled to machine 1 conform to t-configuration r_1, follows similarly. It is to check whether $\sum_{z^\tau \in V} z_k^\tau \leq r_{1,k}$ for all $k \in \{1, \ldots, L+1\}$.

Because there are at most mL/δ big vectors and at most $(\log_{1+\eta} 1/\delta^2)^{L+1}$ different big vectors and since $1/(\delta\eta)$ asymptotically dominates $(\log_{1+\eta} 1/\delta)^{L+1}/\log(1+\eta)$ for small enough δ, the inner DP has at most $O(m^{O(1/\eta\delta)})$ different states:

$$\begin{aligned}
m \cdot \left(\frac{mL}{\delta}\right)^{(\log_{(1+\eta)}(1/\delta^2))^{L+1}} &\leq m^{O\left((\log_{1+\eta} 1/\delta^2)^{L+1} \cdot \log(1/\delta) \cdot \log(L)\right)} && \text{(for } m \geq 3\text{)} \\
&= m^{O\left((\log_{1+\eta} 1/\delta^2)^{L+1} \cdot \log_{1+\eta}(1/\delta)\,/\,\log(1+\eta)\right)} && \text{(as } L \text{ is constant)} \\
&= m^{O\left((\log_{1+\eta} 1/\delta)^{L+2}\,/\,\log(1+\eta)\right)} \\
&= m^{O\left((\log_{1+\eta} 1/\delta)^{L+2}/\eta\right)} = O\left(m^{O\left(\frac{1}{\delta\eta}\right)}\right).
\end{aligned}$$

The number of states is also an upper bound on the time needed to iterate through all different ways to schedule a single machine. The running time is therefore $O\left(m^{O(1/(\delta\eta))}\right)$. □

The following lemma treats the feasibility question of scheduling the small vectors in Z_t^{small} to the t-profile R^{small}. The small vectors are scheduled to the machines by means of an integer linear program formulation and its relaxation. Since the vectors are small, the error induced by rounding the fractional solution to an assignment is only minor.

Lemma 5.4.8 (Chekuri and Khanna [29])**.** *Given a set Z_t^{small} of $(L+1)$-dimensional* small *vectors and a t-profile R^{small}. There is an algorithm that decides in polynomial time whether there exists a schedule for the vectors Z_t^{small} that conforms to t-profile $R^{small} + (\eta, \ldots, \eta, (L+1)\delta)$ or outputs that there is no schedule conforming to t-profile R^{small}.*

Proof. Let the t-profile R^{small} be identified by the vector $(r_1, r_2, \ldots, r_m)$ of t-configurations. Define an integer linear program where $x_i^\tau = 1$ if vector z^τ is assigned to machine i and $x_i^\tau = 0$ otherwise. The algorithm solves the relaxation of the following integer linear program:

$$\sum_{z^\tau \in Z_t^{\text{small}}} x_i^\tau \cdot z_k^\tau \leq r_{i,k} \qquad \forall i \in M,\, k \in [L+1],$$

$$\sum_{i=1}^{m} x_i^\tau = 1 \qquad \forall z^\tau \in Z_t^{\text{small}},$$
$$x_i^\tau \in \{0,1\} \qquad \forall i \in [m],\, z^\tau \in Z_t^{\text{small}}.$$

By classical polyhedral theory, in an extreme point solution there are at most $m(L+1)+n$ positive variables because there are $m(L+1)+n$ constraints, see also Corollary 4.2.9. I show that there are at most $m(L+1)$ vectors that are fractionally assigned to machines. Suppose by contradiction that $m(L+1)+1$ vectors are assigned to multiple machines. Then there are $n-(m(L+1)+1)$ vectors assigned to precisely one machine and $m(L+1)+1$ vectors assigned to at least two machines, which implies that there are at least $n-(m(L+1)+1)+2m(L+1)+2 = n+m(L+1)+1$ positive variables, which contradicts the fact that there are at most $n+m(L+1)$ positive variables. Hence, it must be the case that there are at most $m(L+1)$ vectors which are fractionally assigned to multiple machines.

Partition all vectors that are partially assigned to at least two machines arbitrarily into m groups of at most $L+1$ vectors and assign one such group to each machine. Because the vectors are small each coordinate is smaller or equal to δ. Thus, the extra load for every machine on each coordinate Incluiding the utilization) is at most $(L+1)\delta \leq \eta$. □

The following lemma combines Lemmas 5.4.7 and 5.4.8, yielding the correctness of Algorithm 3 and additionally stating its running time.

Lemma 5.4.9. *Given a set Z_t of $(L+1)$-dimensional vectors and a t-profile R, Algorithm 3 decides in $O(m^{O(C)})$ time whether there exists a schedule for the vectors in Z_t conforming to profile $R+(2\eta,\ldots,2\eta,(L+2)\delta)$ or that there exists no schedule for the vectors in Z_t conforming to profile R.*

Proof. Since the t-profiles are tracked with accuracy η in the first L coordinates and with accuracy δ in their utilization, there are no more than m^C different t-profiles, see Property 5.4.2. Therefore, there are at most m^{2C} combinations of R^{small} and R^{big} that can be combined into the profile R. Lemma 5.4.10, which follows in the next subsection, shows that it takes $O(m^{O(C)})$ time to determine whether two t-profiles R^{small} and R^{big} can be combined into the single t-profile R. Thus, in total, it takes at most $m^{2C}O(m^{O(C)}) = O(m^{O(C)})$ time to split the t-profile R into the t-profiles R^{small} and R^{big}.

Solving the linear program in step 5 takes polynomial time and scheduling the big vectors in step 4 takes $O\left(m^{O\left(\frac{1}{\delta\eta}\right)}\right)$ time, see Lemma 5.4.7. The incurred error follows immediately from Lemma 5.4.8 and the addition of $(\eta,\ldots,\eta,\delta)$ to R^{big} in step 4 of Algorithm 3. □

5.4.6 Splitting t-profiles

Recall that a t-configuration $f = (f_1,\ldots,f_L,f_u)$ specifies how much space might be used by the vectors not scheduled yet on a certain machine in the coordinates t to $t+L-1$, and in the utilization coordinate $K+1$. The inner DP wants to split a t-configuration into two other t-configurations, one specifying the space the big vectors being scheduling might use, and the other specifying the space which all remaining unscheduled big vectors in Z_t^{big} might use, per machine and per relevant coordinate. The sliding window DP on the other hand, wants to

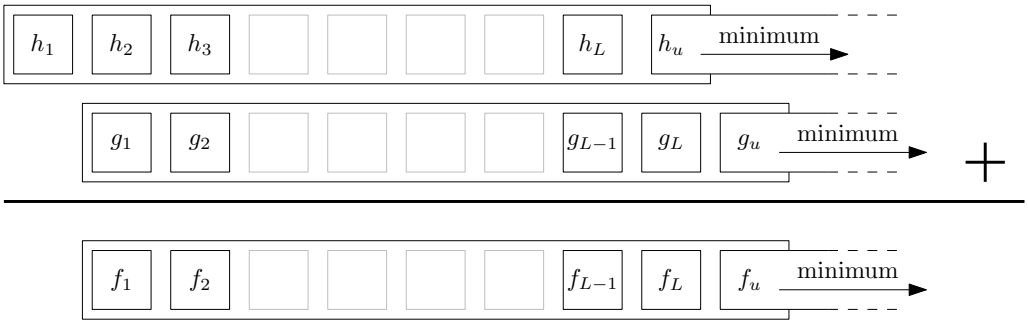

Figure 5.1: Each t-configuration $f = (f_1, \ldots, f_L, f_u)$ encodes on a single machine the space which might be used by vectors not yet scheduled in the coordinates $k \in \{t, \ldots, t + L - 1\}$ and in the coordinate $K + 1$. The figure illustrates how a t-configuration can be split in another t-configuration and a $(t-1)$-configuration.

split a t-configuration into another t-configuration and a $(t-1)$-configuration, specifying the space the vectors in Z_t might use and the space the vectors in $\cup_{k=1}^{t-1} Z_k$ might use, respectively, per machine and per relevant coordinate.

A t-configuration $f = (f_1, \ldots, f_L, f_u)$ can be split into two different t-configurations g and h if and only if

- $f_k \geq g_k + h_k$ for all $1 \leq k \leq L$;
- and $f_u \geq g_u + h_u$.

Further, a t-configuration $f = (f_1, \ldots, f_L, f_u)$ can be split into a t-configuration g and a $(t-1)$-configuration h if and only if

- $f_k \geq g_k + h_{k+1}$ for all $1 \leq k \leq L - 1$;
- and $f_L \geq g_L + h_u$;
- and $f_u \geq g_u + h_u$.

The splitting of a t-configuration into another t-configuration and a lower $(t-1)$-configuration is depicted in Figure 5.1.

Similarly, as t-configurations are split, I split t-profiles. A t-profile $Q = (q_1, \ldots, q_m)$ can be split into a t-profile $R = (r_1, \ldots, r_m)$ and a t-profile $S = (s_1, \ldots, s_m)$ if and only if there are permutations $\alpha, \beta : [m] \to [m]$ such that the t-configuration q_i can be split into the t-configuration $r_{\alpha(i)}$ and t-configuration $s_{\beta(i)}$. In the same way a t-profile can be split into a t-profile and a $(t-1)$-profile.

Lemma 5.4.10. *Determining whether a t-profile Q can be split into a t-profile R and a $(t-1)$-profile S can be done in $O(m^{O(C)})$ time.*

Proof. The goal is to pair configurations from R and configurations from S such that each pair can be combined into a unique configuration from Q. I use the fact that there are at most m^C different profiles because there are C different configurations and a profile can be

alternatively denoted as the tuple $\langle n_1, \ldots, n_C \rangle$ where n_f is the number of machines conforming to configuration f. I describe a simple dynamic programming approach which keeps track of configurations that are not yet paired, therefore there are at most $m^{O(C)}$ states. In each state the DP needs to find a configuration from Q that can be split into a configuration from R and a configuration from S, and remove these configurations. This can done in $O(m^{O(C)})$ time for each state by considering all possibilities. This leads to an algorithm that runs in $O(m^{O(C)})$ time. □

5.5 The $(1+\epsilon)$-feasibility test

Combining Theorem 5.3.4 and Theorem 5.4.1 yields the desired result.

Theorem 5.5.1. *Given $\epsilon > 0$ and task set T, there is an algorithm which correctly decides in $O\left(m^{O(f(\epsilon)\cdot\log(\lambda))}\right)$ time whether (i) T can be partitioned yielding a feasibly assignment $\mathcal{T}$ with speedup factor of $(1+\epsilon)$, or whether (ii) no feasible partition exists in case the machines run at unit speed, where $\lambda = d_n/d_1$ and $f(\epsilon)$ is a function depending solely on ϵ.*

Proof. Theorem 5.4.1 determines in $O(m^{O(C)})$ time whether a feasible solution to the vector scheduling problem exists with a speedup factor of $1+\epsilon$, or whether no such partition of the vectors to the machines exists without speedup. Thus in light of Theorem 5.4.1, Theorem 5.3.4 implies that (i) if there exists a feasible partition for the vector scheduling problem, then this partition is feasible for the realtime scheduling problem is the machines receive a speedup factor of $(1+2\epsilon)^3(1+\epsilon)$, and that (ii) if no feasible partition for the vector scheduling problem exists, then no feasible partition exists for the realtime scheduling problem in case the machines run at speed $1/(1+\epsilon)$. Rescaling ϵ appropriately yields the stated result. □

5.6 Smoothing in realtime scheduling

In this final section I turn to applying the concept of smoothing to a realtime scheduling problem. I motivated the concept of smoothing in Chapter 3 as a technique which tries to bring theoretical performance bounds closer to the behavior of algorithms observed in practice. Often simple algorithms are powerful in practice but might have bad theoretical performance due to some rare artificial instances which one would not expect to arise in practical applications. Such behavior is also observed for the Earliest Deadline First (EDF) algorithm and similar policies. In this section I study the running time for the rate monotonic algorithm for scheduling sporadic tasks with implicit deadlines on a single machine. Thus, the studied problem is a slightly simpler problem than the *co-$\mathcal{NP}$-hard* feasibility problem of EDF scheduling sporadic tasks with constrained deadlines on a single machine, [43]. In worst case though, the studied algorithm might still take pseudo-polynomial time to decide whether a task system with implicit deadlines is feasible on a single machine. I show that in the presence of smoothing the situation is more promising as the expected running time of the rate monotonic algorithm then is polynomial in the number of tasks and only linearly depends on the smoothing parameter.

As opposed to the previous sections of this chapter, I wrote this section on my own account being inspired by lively research discussions with Alberto Marchetti-Spaccamela, Suzanne

van der Ster and Andreas Wiese, and by a minicourse on smoothed analysis by Heiko Röglin which is based upon [17].

5.6.1 Problem definition and a dynamic program

I consider the realtime scheduling problem of determining whether a system of sporadic tasks with implicit deadlines can be feasibly scheduled on a single machine. I assume that the tasks arrive according the synchronous arrival sequence. Recall from Subsection 1.2.3 that tasks having implicit deadlines implies that $d_\tau = p_\tau$ for all tasks τ. Hence, I will drop the deadlines in the remainder of the section and speak of tasks' periods only.

Though the final result which will be presented can be extended to EDF to yield a true PTAS in expectation, I will show the result for a run-time algorithm which is actually slighty different from EDF. I will consider *fixed priority scheduling* where each task is given a (offline) priority. The processor will process at each point in the time the pending job with highest priority among all pending jobs. The *Rate Monotonic algorithm* (RM) assigns to each task τ the priority $1/p_\tau$, that is, among all pending jobs, the processor will process the job with smallest period first (which need not be the job with the closest deadline). RM is *optimal* in the sense that if any priority algorithm finds a feasible schedule, then RM will also find a feasible schedule. Note that RM might be worse than EDF though as EDF does not present a fixed priority scheduling policy. The reason I study the rate monotonic algorithm here is that the algorithm is preferred in practice over EDF as it is simpler to implement: the decision which job needs to be processed next is independent of the moment in time at which the decision needs to be made (which is not true for EDF). I will assume that tasks are ordered to their priority, i. e., $p_1 \leq p_2 \leq \ldots \leq p_n$. As tasks are ordered, I will in the remainder refer to a task by the index $j \in \{1, \ldots, n\}$. A property similar to second condition of Property 4.2.6 is known for defining the feasibility of RM and implies that for deciding upon feasibility only the first job of each task really matters:

Property 5.6.1. *An implicit deadline system is feasible for the rate monotonic scheduling algorithm on a single machine if and only if in the synchronous arrival sequence each task's first job meets its deadline (which is equal to p_j).*

The *response time* r_j of a task j is the first moment in time at which it is provable that the first job of task j is scheduable. The response time r_j for a tasks j scheduled according to rate monotonic schedule and the synchronous arrival sequence, is defined to be the minimum moment in time $r_j > 0$ for which

$$r_j = c_j + \sum_{k \in T : p_k < p_j} \left\lceil \frac{r_j}{p_k} \right\rceil c_k.$$

Hence, testing for feasibility of RM can be rephrased by testing whether $r_j \leq p_j$ for all tasks $j \in T$. Define the following function for task $j \in \{1, \ldots, n\}$:

$$f_j(r) = c_j + \sum_{k \in T : p_k < p_j} \left\lceil \frac{r}{p_k} \right\rceil c_k.$$

The above definition gives rise to the following property which is, among others, also presented in [42]:

Property 5.6.2. *A task system T of sporadic tasks with implicit deadlines can be feasibly scheduled up to task j using the rate monotonic algorithm if and only if,*

1. *the task system is feasible up to task $j-1$, and*
2. *if there exists a time point $r \leq p_j$ such that $f_j(r) \leq r$.*

The preceding property yields a pseudo-polynomial test. Assume tasks' periods to be positive integers. Then one could simply check, for task j, whether $f_j(r) \leq r$ for some $r \in \{1, 2, \ldots, p_j\}$. If such a time point r is found then the algorithm returns 'FEASIBLE' up to task j and otherwise 'INFEASIBLE'. The dynamic program iterates this procedure until either 'INFEASIBLE' is returned for some task, or until the final task n returns 'FEASIBLE'. The dynamic programming algorithm based upon the previous discussion is given by Algorithm 4. The algorithm does however not need to compute $f_j(r)$ for all possible values of $r \in \{1, 2, \ldots, p_j\}$, see the following lemma:

Lemma 5.6.3. *If $f_j(r) > r$, then $f_j(s) > s$ for all $s \in [r; f_j(r) - 1]$.*

Proof. The function $f_j(r) = c_j + \sum_{k:p_k<p_j} \left\lceil \frac{r}{p_k} \right\rceil c_k$ is nondecreasing in r. Hence, $f_j(s) \geq f_j(r) > s$ for all $s \in [r; f_j(r) - 1]$. □

Algorithm 4 RM dynamic programming version

Require: Given is a set of tasks T s.t. $p_1 \leq \ldots \leq p_n$, and where p_j is integer for all tasks j

```
For j = 1 to n
  Set TaskScheduable = False
  r = 1
  While r ≤ p_j And !TaskScheduable Do
    If f_j(r) ≤ r
      TaskScheduable = True
    Otherwise
      r = ⌈f_j(r)⌉
    End If
  Loop
  If !TaskScheduable Then Return Infeasible
Next j
Return Feasible.
```

Given that the task system is feasible up to task $j-1$, the way Algorithm 4 proceeds to check whether the system is feasible up to task j is illustrated in Figure 5.2.

The algorithm takes the same steps as the function $f_j(r)$ does. Hence, the running time of the algorithm is bounded by number of different values the function $f_j(r)$ might take over the range $r \in [1, p_j]$.

Proposition 5.6.4. *Let x_j denote the number of different values $f_j(r)$ might take over the range $r \in [0, p_j]$. The running time of the dynamic programming algorithm for phase j is $O(x_j)$.*

As the periods are assumed to be integer, the number of different values the function $f_j(r)$ might take over the range $r \in [0, p_j]$ is no more than p_j.

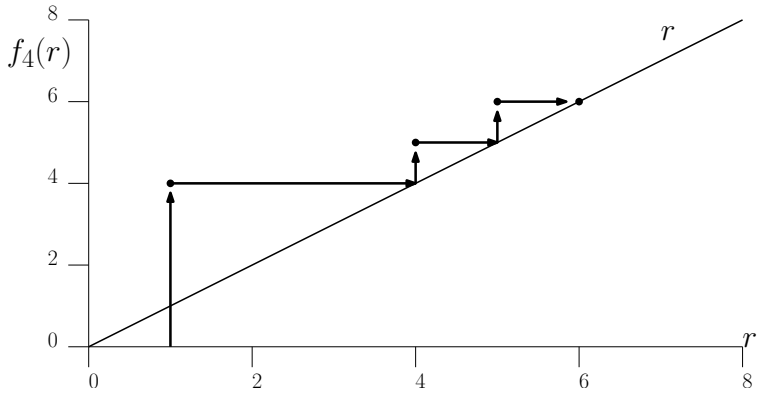

Figure 5.2: Progress of Dynamic Programming Algorithm on an instance consisting of four tasks where $(c_1, p_1) = (1, 3)$, $(c_2, p_2) = (1, 4)$, $(c_3, p_3) = (1, 8)$ and $(c_4, p_4) = (1, 10)$. Consider scheduling the fourth task. Then, $f_4(1) = 4$, $f_4(4) = 5$, $f_4(5) = 6$ and finally $f_4(6) = 6$ such that there exists a point in time $r = 6 \leq p_4$ where $f_4(r) \leq r$.

Proposition 5.6.5. *The running time of Algorithm 4 is in* $O(\sum_{j \in T} p_j)$.

Lemma 5.6.6. *The running time of Algorithm 4 is in* $\Omega(\sum_{j \in T} p_j)$.

Proof. I will provide an example showing the lower bound. Let $\epsilon < 2^{-n}$ be an appropriate small constant. Consider n tasks such that $(c_1, p_1) = (1 + \epsilon, 2)$ and $(c_j, p_j) = (1, 2^j)$ for all tasks $j \in \{2, \ldots, n\}$. The response time r_j of tasks j then equals $2^j - 1 + 2^{j-1}\epsilon > p_j - 1$. One can validate that the number different values the function $f_j(r)$ takes up to time point p_j, denoted by $\#f_j$, is given by the recursive formula

$$\#f_j = 2 \cdot \#f_{j-1} - j + 2, \quad \text{where } \#f_2 = 3.$$

That is, $\#f_j \approx 2 \cdot \#f_{j-1}$, whereas $p_j = 2 \cdot p_{j-1}$ for $j > 2$. Therefore, $\#f_j = \Omega(p_j)$ such that the total running time of n iterations is in $\Omega(\sum_{j \in T} p_j)$. □

5.6.2 Smoothing the processing requirements

I show that if the processing times of tasks are being smoothed, then the number of distinct values the function $f_j(r)$ can take over the range $r \in [1; p_j]$ is bounded by the smoothing parameter ϕ. Therefore, the smoothed running time of Algorithm 4 is polynomial in the number of tasks and the smoothing parameter ϕ.

Let X_j denote the number of different values the function $f_j(r)$ can take over the range $r \in [1; p_j]$. Since the scheduling problem considered is invariant to scaling, that is, when all c_j and all p_j are scaled by the same constant, I may assume in the remainder that $f_n(p_n) \leq 1$ by dividing all periods and processing times by $f_n(p_n)$. The latter assumption implies that $p_j \leq 1$ (such that $c_j \leq 1$) for all tasks $j \in T$ (as otherwise in constant time it can be validated that $r = p_n \geq f_n(p_n)$ shows feasibility).

Let me now introduce the smoothing model. I specify the smoothing model as follows: let χ_j be the random variable for the processing time of task j. Further, χ_j is drawn from the

cumulative distribution function $\Phi_j(a) = \mathbf{Pr}[\chi_j \leq a]$ for any constant $a \in [0,1]$. Assume that the function $\Phi_j(a)$ is differentiable in a over its full range and let $\varphi_j(a) = \frac{\partial}{\partial a}\Phi_j(a)$ denote the resulting probability density function of the variable χ_j. I restrict the power of the adversary by assuming that $\varphi_j(a) \leq \phi$ for any value of a. Here, $\phi \geq 1$ denotes the smoothing parameter and, as in Chapter 3, ϕ forms an upper bound on the value the probability density function of the random variable χ_j can take. I will refer to a task system where the processing times are generated in this way as a *ϕ-smooth task system*. To illustrate an example of a ϕ-smooth task system, consider for instance the following additive smoothing model:

$$\chi_j = \begin{cases} \frac{1}{2\phi} + \epsilon_j & \text{if } c_j < \frac{1}{2\phi}, \\ c_j + \epsilon_j \text{ if} & \frac{1}{2\phi} \leq c_j \leq 1 - \frac{1}{2\phi}, \\ 1 - \frac{1}{2\phi} + \epsilon_j & \text{otherwise}, \end{cases}$$

where ϵ_j are independent random variables all drawn from a uniform distribution over the range $[-\frac{1}{2\phi}, \frac{1}{2\phi}]$, for all tasks $j \in T$.

The term ϵ_j might be interpret as the noise on the processing times of the original instance. This noise models the disturbances on the input of an algorithm due to imprecise measurements such as rounding errors or numerical imprecision. The smoothing parameter ϕ controls how much noise might be present. For instance, in the additive smoothing model, if ϕ is close to one, then the smoothed instance is close a random instance generated with an uniform distribution over the range zero to one. Alternatively, if ϕ is arbitrary large, then the noise added to the instance is arbitrary small and negligible, and hence, we are back in worst case analysis. Hence, smoothing might be seen as a hybrid between average case and worst case analysis.

The main theorem of this section is given next and is followed by the corollary establishing the smoothed running time of the rate monotonic algorithm.

Theorem 5.6.7. *Consider a ϕ-smooth task system where $p_1 \leq \ldots \leq p_j$ represent the periods and $\chi_1, \ldots, \chi_j$ the smoothed processing times. Let X_j denote the number of distinct values the function $f_j(r)$ might take over the range $r \in [0, p_j]$. Then $\mathbf{E}[X_j] \leq \phi$.*

The previous theorem together with Proposition 5.6.4 yield that the expected running time of Algorithm 4 is polynomial in the number of tasks and the smoothing parameter ϕ.

Corollary 5.6.8. *On a ϕ-smooth task system, the expected running time of the rate monotonic algorithm as described in Algorithm 4 is $O(n\phi)$.*

I present the proof of the theorem below. An insight of probability theory, which is being used in the proof of this theorem, is shown and proven afterwards.

Proof of Theorem 5.6.7. The realtime scheduling problem presented is invariant to scaling. Therefore, assume without loss of generality that $f_n(p_n) \leq 1$ (by dividing all the original χ_j and p_j by $f_n(p_n)$). Since $\chi_j > 0$ for all tasks $j \in T$, I can partition the range $[0,1]$ in $(1/\delta)$ slices of height δ. By choosing δ small enough it is established that each slice contains at most one distinct value of $f_j(r)$ (with arbitrary high probability). The slicing is illustrated in Figure 5.3. Let slice k cover the range $(\,(k-1)\delta;\, k\delta\,]$. Let the random variable $Z_k \in \{0,1\}$ denote whether

the function $f_j(r)$ takes a value in slice k. Note that $f_j(0) = c_j > 0$ and hence time 0 can never be the response time for any task j. Then,

$$\mathbf{E}[X_j] = \lim_{\delta \downarrow 0} \sum_{k=1}^{1/\delta} \mathbf{E}[Z_k] = \lim_{\delta \downarrow 0} \sum_{k=1}^{1/\delta} \mathbf{Pr}[Z_k = 1]\,.$$

Let $r_k = \min\{r \in [0, p_j] : f_j(r) > (k-1)\delta\}$. Thus, for all $r < r_k$ it holds that $f_j(r) \leq (k-1)\delta$ whereas for all $r \geq r_k$ it needs to be that $f_j(r) > (k-1)\delta$ (note $f_j(r)$ is nondecreasing in r). The latter implies that at time moment r_k some jobs are being released, let me refer to these jobs as job set J. Then,

$$\mathbf{Pr}[Z_k = 1] = \mathbf{Pr}[f_n(r_k) \in ((k-1)\delta, k\delta]\,] = \mathbf{Pr}\left[\sum_{j \in J} \chi_j \in \left((k-1)\delta - f^<, k\delta - f^<\right]\right],$$

where $f^<$ is the value the function $f_n(r)$ took right before time moment r_k. By Lemma 5.6.9 (which follows below) it follows that

$$\mathbf{Pr}[Z_k = 1] = \mathbf{Pr}\left[\sum_{j \in J} \chi_j \in \left((k-1)\delta - f^<;\, k\delta - f^<\right]\right] \leq \phi\delta.$$

Hence,

$$\mathbf{E}[X_j] = \lim_{\delta \downarrow 0} \sum_{k=1}^{1/\delta} \mathbf{Pr}[Z_k = 1] \leq \lim_{\delta \downarrow 0} \sum_{k=1}^{1/\delta} (\phi\delta) = \phi.$$

□

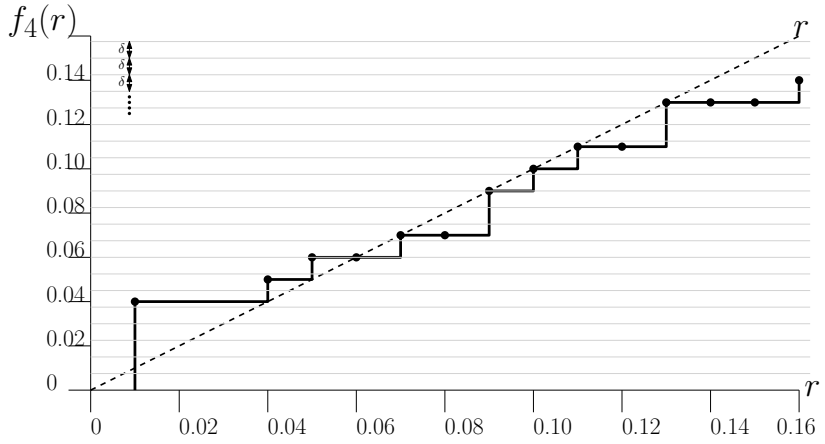

Figure 5.3: The function of $f_4(r)$ depicted for a system of four tasks, where $(c_1, p_1) = (0.01, 0.03)$, $(c_2, p_2) = (0.01, 0.04)$, $(c_3, p_3) = (0.01, 0.08)$ and $(c_4, p_4) = (0.01, 0.10)$. The vertical axis has been partitioned into slices of height $\delta = 0.0075$ such that the function $f_4(r)$ takes at most one unique value within a slice.

5.6.3 An insight from probability theory

The following lemma bounds the height of the probability density function of accumulated random variables. The lemma was used in the proof of Theorem 5.6.7. Let $Y_1, \ldots, Y_n$ be a collection of independent random variables for $n \geq 2$, and assume that their cumulative probability density functions $\Phi_j(a) = \mathbf{Pr}[Y_j \leq a]$ are differentiable over their full range such that $\varphi_j(a) = \frac{\partial}{\partial a}\Phi_j(a)$, for $j \in \{1, \ldots, n\}$, denote their probability density functions. Further, define $W_n = \sum_{j=1}^{n} Y_j$ to be the accumulated random variable and let $\varphi_{W_n}(a)$ denote the corresponding probability density function (which exists as the functions $\varphi_j(a)$ are assumed to exist).

Lemma 5.6.9. *Let $Y_1, \ldots, Y_n$ be a collection of independent random variables such that $\varphi_j(a) \leq \phi$ for any constant a and all $j \in \{1, \ldots, n\}$, that is, the probability density function of Y_j is upper bounded by ϕ. Then it holds that $\varphi_{W_n}(a) \leq \phi$ for any constant a, that is, the probability density function of the sum of the random variables is also upper bounded by ϕ.*

Proof. Since $\varphi_j(a) \leq \phi$, for $j \in \{1, \ldots, n\}$, represent probability density functions, it follows that these functions satisfy:

$$\varphi_j(\cdot) : (-\infty, +\infty) \to [0, \phi] \tag{5.7}$$

and

$$\int_{-\infty}^{+\infty} \varphi_j(a)\, \partial\, a \;=\; 1. \tag{5.8}$$

By induction, assume that $\varphi_{W_{n-1}}(a) \leq \phi$ for any constant a. I will show that then $\varphi_{W_n}(a) \leq \phi$ for any constant a. Consider any value a, then

$$\begin{aligned}
\varphi_{W_n}(a) &= \int_{-\infty}^{+\infty} \varphi_{W_{n-1}}(y) \cdot \varphi_n(a-y)\, \partial y && \text{by independence} \\
&\leq \int_{-\infty}^{+\infty} \phi \cdot \varphi_n(a-y)\, \partial y && \text{by induction} \\
&= \phi \cdot \int_{-\infty}^{+\infty} \varphi_n(z) \partial z && \text{by change of variable} \\
&= \phi. && \text{due to (5.8) for } Y_n
\end{aligned}$$

The first induction step follows since for $n = 2$, $W_{n-1} = W_1 = Y_1$ such that $\varphi_{W_1}(a) = \varphi_1(a) \leq \phi$ for all a, see Equation (5.7). □

Chapter 6

A powerful learning policy in stochastic scheduling

This chapters considers a scheduling problem in which m classes of independent jobs have to be processed non-preemptively by a single machine. The processing times of the jobs are assumed to be exponentially distributed with parameters depending on the class of each job. The objective is to minimize the sum of expected completion times. I adopt a Bayesian framework in which the parameters of the job classes are assumed to be unknown. However, by processing jobs from a specific class, the scheduler can gradually learn about the value of the corresponding parameters, thereby enhancing his decision making in the future.

An optimal dynamic program for the considered problem has been designed [63]. However, as this program is highly impractical in both implementation and running time, powerful and simple policies are needed for practical purposes. In this chapter two such greedy policies are studied. For the traditional stochastic scheduling variant, in which the parameters are known, the policy that always processes a job with Shortest Expected Processing Time (SEPT) is optimal. In this new problem with parameter uncertainty, I show that in the Bayesian framework the performance of SEPT is at most a factor m away from the performance of an optimal policy. Further, instances are given for which this bound is tight. Furthermore, I introduce a second policy called learning-SEPT (ℓ-SEPT), which is an adaptive variant of SEPT. I show that ℓ-SEPT is no worse than SEPT, and has a smaller lower bound of $1 + \sqrt{m-1}$ on the performance guarantee. Moreover, it is conjectured that this lower bound is actually the correct performance guarantee for ℓ-SEPT. Finally, I provide computational results showing that ℓ-SEPT empirically outperforms SEPT. Further, as a special case, I comment on the case where there are just two job classes.

This chapter is the result of joint research with Sebastián Marbán and Tjark Vredeveld [86].

6.1 Introduction

Problem definition. This chapter considers a variation of the traditional stochastic problem which was described in Subsection 1.2.3. The objective is to non-preemptively schedule jobs belonging to different classes on a single machine in order to minimize the sum of com-

pletion times. Processing requirements are assumed to be stochastic. In traditional stochastic scheduling, this problem is well understood and can be solved to optimality by the Shortest Expected Processing Time (SEPT) policy, see Subsection 1.3.1 and [100, 111]. In almost all work in stochastic scheduling, jobs' processing requirements are independent random variables of which the parameters, like the expected value, are fully known. In the current chapter, I relax this assumption by introducing *parameter uncertainty*. Like in [26, 57, 59, 63, 64], I adopt a *Bayesian* viewpoint in which prior distributions are given for the uncertain parameters. These priors represent the scheduler's beliefs on the values of these parameters. Furthermore, the Bayesian framework allows to learn about the value of the parameters by processing jobs and observing their realized processing times. However, experimenting with different jobs to learn about the value of the corresponding parameters can be costly in terms of the waiting times of the still to be processed jobs. Hence, learning should be conducted carefully in order to minimize the sum of completion times in expectation.

The motivation for studying this problem stems from the believe that the traditional and well-studied stochastic scheduling problem is not realistic enough when the production conditions are subject to change. Consider for example the production line of a car manufacturer. Say, she is already producing two types of cars, A and B, on the given production line and she wants to add a new third type of car, type C. Since she is already producing types A and B for a long time, she knows exactly how the distributions of the production time of those two types look like, i. e., she knows all the parameters of the distributions. The third type she wants to start producing is new however, and therefore I believe that it is unrealistic to assume the car manufacturer knows the exact properties of the distribution on the production times. Instead, we find it plausible that she has some initial ideas on the production time distribution stemming from her experience as she has introduced many car types before, but she does not know its exact shape. However, once she starts producing type C, she will learn of those distribution parameters. Moreover, the more type C cars she has produced, the more sure she will be about the production time distribution.

I proceed with describing the problem setting introduced above formally. There are m classes of independent jobs which have to be processed by a single machine. Each class J_i consists of n_i jobs, with $i \in \{1, \ldots, m\}$. All jobs are available for processing from the beginning and preemption of jobs is not allowed. The processing time of a job in class J_i is a random variable, which is independently and exponentially distributed with parameter ϑ_i. The goal is to minimize the total completion time in expectation, $\sum_j \mathbf{E}[C_j]$. Distinguishing from traditional stochastic scheduling, in the scheduling model under consideration the value of ϑ_i is unknown. The random variable Θ_i captures the scheduler's beliefs regarding the value of ϑ_i. In the Bayesian approach, mathematically speaking, ϑ_i can be considered as a realization of the random variable Θ_i. The initial distribution of Θ_i, that is, before any job has been processed, is called the *prior* (distribution). As in [63, 64], it is assumed that the prior is a gamma distribution with parameters $\omega_i > 0$ and $\alpha_i > 1$.

The gamma distribution is a suitable choice since unlike some other commonly used distributions, the support of the gamma is nonnegative. Further, depending on the confidence in his beliefs about ϑ_i, the scheduler can choose the values of ω_i and α_i such that the prior is very peaked (the scheduler is very certain about his beliefs) or relatively flat (the scheduler is not certain about his beliefs) or anywhere in between. Note also that the popular normal distribution with parameters μ and σ can be fitted by a gamma distribution by setting $\alpha = \mu^2/\sigma^2$

and $\omega = \mu/\sigma^2$ when α is sufficiently large.

After a job of class J_i is processed, this job's processing time x is revealed. Since the gamma distribution presents a conjugate prior for the exponential distribution, the *posterior* distribution of Θ_i, representing the beliefs of ϑ_i after having observed processing time realization x, is a (more peaked) gamma distribution with parameters $\omega_i + x$ and $\alpha_i + 1$. This result is stated in, among others, [40] and is also derived from Bayes' theorem for probability density functions. In this way, the scheduler gradually learns about the unknown parameter, thereby enhancing his decision making in the future.

In this scheduling problem with uncertainty, I define an *optimal scheduling policy* as a non-anticipatory scheduling policy which minimizes the objective value in expectation. Note hereby that an optimal scheduling policy underlies the uncertainty about processing times as well as the uncertainty about the parameters. To be more explicit, the optimal policy realizes that certain scheduling decisions will reveal additional information in the near future. However, as an optimal policy is non-anticipatory, it does not know how that information is going to look like. Burnetas and Katehakis [26] and Hamada and Glazebrook [63] present such optimal policies for different number of job classes. Even for the case of two job classes, one of which has known parameter, these policies require solving an extensive dynamic program, where in each iteration of the program a vast amount of non-linear equations needs to be solved with high accuracy. As a result, solving instances of the Bayesian scheduling problem to optimality is highly impractical, if not impossible. Thus, for practical purposes, there is a need for simple and powerful policies which return good schedules fast.

SEPT constitutes a straightforward candidate for such a policy. In the traditional stochastic scheduling variant of the problem studied here, the simple SEPT poses an optimal scheduling policy [100]. In the Bayesian setting, there are two natural versions of SEPT. The first one, which I will keep calling SEPT, determines the order in which the jobs will be processed at the beginning based on its initial beliefs. The second version, which we denote by *learning*-SEPT or ℓ-SEPT is dynamic and updates its beliefs on ϑ_i every time a job of class J_i is completed. After each completion of a job, ℓ-SEPT will schedule the job with shortest expected processing time with respect to its current beliefs. In this chapter, I investigate the quality of the solution value obtained by both policies.

Related work. The deterministic version of the scheduling problem studied can be solved to optimality by the SPT rule [111]. When preemption is allowed, the problem is solved by the Shortest Remaining Processing Time (SRPT) rule [107]. On more than one machine, the performance guarantee of the SRPT rule has been studied in [96] and subsequently improved in [34] and [110].

In traditional stochastic scheduling, the processing times of jobs are random variables for which the parameters of the underlying distribution are known. Rothkopf [100] shows that Weighted Shortest Expected Processing Time (WSEPT) is an optimal policy for the stochastic single machine scheduling problem, where the objective is to minimize the sum of weighted expected completion times. Weiss [117, 118] analyzes the performance of WSEPT for the stochastic parallel machines scheduling problem. He shows asymptotic optimality of WSEPT for a certain class of processing time distributions. The first guarantee on the quality of an approximative policy was given by Möhring, Schulz, and Uetz [93]. Other approximative

policies have been considered in [39, 88, 89, 108].

The current chapter combines parameter uncertainty with stochastic scheduling and applies a Bayesian framework to model this parameter uncertainty. Examples of papers which apply such a Bayesian framework to scheduling problems are limited. In the pioneering paper of Gittins and Glazebrook [57], the distributions of processing times of jobs depend all upon the same unknown parameter. The optimal schedule is obtained by calculating appropriate dynamic allocation indices, first proposed by Gittins and Jones [58]. Hamada and Glazebrook [63] present another example studying the Bayesian scheduling problem with multiple weighted job classes. The processing time distributions of these classes depend on either known or unknown parameters. Optimal policies are derived by formulating the problem as a dynamic program and using dynamic allocation indices similar to the ones in [55, 56]. Burnetas and Katehakis [26] derive dynamic programming optimality conditions for the same problem with two classes: one with known and one with unknown underlying parameter. For arbitrary processing time distributions, they show that an optimal policy does not depend on the number of jobs in the class with known parameter. As a result, the optimality conditions simplify, allowing the problem to be modeled as a stopping problem. For exponential processing times, they thereafter obtain characterizations concerning the structure and properties of the optimal policy. Finally, a policy is given that approximates the decisions made by an optimal policy if the number of jobs in the class with unknown parameter tends to infinity. There are few more papers which combine stochastic scheduling with Bayesian methodology. In [59], Glazebrook and Owen employ a comparative study in which they seek to quantify the difference between using adaptive scheduling policies based on Bayesian methodology and non-adaptive classical stochastic scheduling policies. The difference is called the 'value of an adaptive solution'. Upper bounds on this value are derived for several scheduling problems on a single machine. Further, Rieder and Weishaupt [99] study a problem in which jobs may change their class with some fixed but unknown probability. A Bayesian approach is taken to update beliefs regarding these probabilities. Finally, the paper of Hamada and Tamaki [64] constitutes the first extension to two machines, for two job classes. Again, a dynamic programming formulation is used to derive optimality conditions.

Bayesian methodology is widely applied in research fields related to scheduling. In inventory management for example, there is a large body of literature dealing with uncertain demand distributions and Bayesian learning. Pioneered by [9, 104], some recent papers are given by [32, 77, 85]. The majority of these papers assumes that prices are exogenous and studies the problem of making optimal inventory decisions. Bayesian demand learning has also received a great deal of attention within the field of pricing, see [4, 36, 82]. All these papers are experimental in that they focus on developing heuristics and studying their computational aspects. The first, and so far only, paper to analyze the theoretical worst-case performance of a Bayesian pricing heuristic is [45].

Contribution. I generalize from traditional stochastic single machine scheduling by assuming that the distribution of the processing requirements needs not to be known with full certainty, i. e., there might be uncertainty around the distribution's parameters. I apply a Bayesian framework to model this parameter uncertainty. Since optimal policies for the studied problem are not suited for practical purposes, I study the performance of two simple and

popular scheduling policies: SEPT and learning-SEPT. First, I show that ℓ-SEPT always outperforms the non-adaptive version SEPT. Secondly, I show that both performance guarantees of SEPT and ℓ-SEPT are upper bounded by m. I show that there exist instances with non-degenerately distributed processing requirements for which this bound is tight for SEPT. For ℓ-SEPT, I show a lower bound on the performance guarantee of $1 + \sqrt{m-1}$ and I conjecture that this is the true performance guarantee. The conjecture is supported by computational results. These empirical findings further strengthen that ℓ-SEPT is a powerful tool for practical purposes as it is shown to be often almost optimal.

Outline. In the next section, I continue with some preliminary insights on Bayesian methodology and creating a more elaborate understanding of the policies SEPT and ℓ-SEPT. Sections 6.3 and 6.4 provide the upper and lower bounds for SEPT and ℓ-SEPT respectively. Section 6.5 provides the conjecture on the true performance guarantee of ℓ-SEPT. The second part of this section presents the computational results supporting the conjecture. Section 6.6 presents the computational results evidencing that ℓ-SEPT (i) significantly outperforms the non-adaptive variant SEPT and (ii) and on the majority of instances is almost optimal. I conclude in Section 6.7 by remarking on the special case of just 2 job classes.

6.2 Preliminaries and scheduling policies

In this section, I will elaborate more on the Bayesian scheduling framework and the policies SEPT, ℓ-SEPT, and OPT. Additionally, I give some preliminary bounds on the performance of these policies.

6.2.1 Bayesian methodology

Bayesian methodology offers a method to formally recognize the uncertainty regarding parameter ϑ_i. A random variable Θ_i is introduced which describes the scheduler's beliefs regarding the value of ϑ_i. In the Bayesian approach, ϑ_i can be considered as a realization of the random variable Θ_i. For some $\theta > 0$, let $g_i(\theta) := \frac{\partial}{\partial\theta}\mathbf{Pr}[\Theta_i \leq \theta]$ denote the *prior* probability density function. Intuitively, the probability $\mathbf{Pr}[\Theta_i \leq \theta]$ expresses how strongly the scheduler believes that the value of ϑ_i is less than or equal to θ, prior to seeing any realization of processing times of jobs of class J_i. As mentioned before, it is assumed that Θ_i follows a gamma distribution with parameters $\omega_i > 0$ and $\alpha_i > 1$. Let X_i^k denote the random variable expressing the processing time of the k^{th} job of job class J_i. Once k jobs of class J_i have been completed with processing time realizations x_i^1 up to x_i^k, the beliefs with respect to the unknown value of ϑ_i will be updated and expressed by the *posterior* probability density function

$$g_i(\theta|x_i^1,\ldots,x_i^k) := \frac{\partial}{\partial\theta}\mathbf{Pr}[\Theta_i \leq \theta|X_i^1 = x_i^1,\ldots,X_i^k = x_i^k]\,.$$

Since the gamma distribution presents a conjugate prior for the exponential distribution, the posterior is also a gamma with parameters $\omega_i' := \omega_i + \sum_{j=1}^k x_i^j$ and $\alpha_i' := \alpha_i + k$ (e. g., see Section 9.4 of [40]).

Updating beliefs towards ϑ_i results in updated beliefs regarding the processing times of uncompleted jobs in class J_i. The posterior probability density function expressing these latter

beliefs, after having completed k jobs of class J_i, is denoted by

$$f_i^k(x_i^{k+1}) := \frac{\partial}{\partial x_i^{k+1}} \mathbf{Pr}[X_i^{k+1} \leq x_i^{k+1} | X_i^1 = x_i^1, \ldots, X_i^k = x_i^k],$$

which is equal to

$$\begin{aligned} f_i^k(x_i^{k+1}) &= \int_0^\infty f(x_i^{k+1}|\theta) g_i(\theta | x_i^1, \ldots, x_i^k) \partial\theta \\ &= \int_0^\infty \theta e^{-\theta x_i^{k+1}} \frac{{\omega_i'}^{\alpha_i'}}{\Gamma(\alpha_i')} \theta^{\alpha_i'-1} e^{-\theta\omega_i'} \partial\theta = \frac{\alpha_i' {\omega_i'}^{\alpha_i'}}{(\omega_i' + x_i^{k+1})^{\alpha_i'+1}}, \end{aligned} \tag{6.1}$$

where $f(x_i^{k+1}|\theta)$ is an exponential probability density function with parameter θ. Furthermore, straightforward integration yields the first moment of X_i^{k+1} given the first k realizations:

$$\mathbf{E}[X_i^{k+1} | X_i^1 = x_i^1, \ldots, X_i^k = x_i^k] = \int_0^\infty x_i^{k+1} f_i^k(x_i^{k+1}) \, \partial x_i^{k+1} = \frac{\omega_i + \sum_{j=1}^k x_i^j}{\alpha_i + k - 1}. \tag{6.2}$$

The more jobs of job class J_i have been processed, the more accurate the scheduler's beliefs regarding ϑ_i will be. First, the expected value of $(\Theta_i | X_i^1 = x_i^1, \ldots, X_i^k = x_i^k)$ will converge to ϑ_i by the law of large numbers. Secondly, the variance of $(\Theta_i | X_i^1 = x_i^1, \ldots, X_i^k = x_i^k)$ will decrease since ω_i and α_i will be increased with every new observation. Hence, the more jobs processed, the more peaked and the more centered around ϑ_i the distribution of $(\Theta_i | X_i^1 = x_i^1, \ldots, X_i^k = x_i^k)$ will become, i. e., the more the scheduler learns about the value of ϑ_i.

6.2.2 Bayesian scheduling policies

An optimal policy for the Bayesian scheduling problem at hand, OPT, minimizes total completion time in expectation, thereby taking into account the uncertainty regarding the job class parameters. That is, the values of the parameters ϑ_i are unknown to OPT, but the policy will anticipate and act in its decision making upon the revealment of additional information when processing a job of a certain job class. In order to characterize OPT, I formulate the problem as a dynamic program, as introduced by [63].

Let $\boldsymbol{n} = (n_1, \ldots, n_m)$, $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_m)$, and $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)$. Then, $(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha}) \in \mathbb{Z}_+^m \times \mathbb{R}_{>0}^m \times \mathbb{R}_{>1}^m$ denotes a state vector encompassing all relevant information of the state the system is in. It consists of the number of jobs in each class J_i as well as the parameters of the current belief for ϑ_i. Let $\boldsymbol{e}_i$ be the i^{th} unit vector. If in state $(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})$, a job of class J_i is processed and completed having realization x, then the state changes to $(\boldsymbol{n} - \boldsymbol{e}_i, \boldsymbol{\omega} + x\boldsymbol{e}_i, \boldsymbol{\alpha} + \boldsymbol{e}_i)$. Let $\mathbf{E}[\Pi^*(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})]$ denote the expected sum of completion times when the optimal policy is adopted from state $(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})$ onwards. Further, let $\mathbf{E}[\Pi_i^*(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})]$ denote the sum of the expected completion times of a policy which first processes a job of class J_i (assuming $n_i \geq 1$) and follows an optimal policy afterwards. An optimal policy OPT can then be modeled by the following dynamic program:

$$\mathbf{E}[\Pi^*(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] = \min_{i \in [m]: n_i \geq 1} \{\mathbf{E}[\Pi_i^*(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})]\} \tag{6.3}$$

and

$$\mathbf{E}[\Pi^*(n_i\boldsymbol{e}_i,\boldsymbol{\omega},\boldsymbol{\alpha})] = \left(\sum_{j=1}^{n_i} j\right)\frac{\omega_i}{\alpha_i-1} = \frac{n_i(n_i+1)}{2}\frac{\omega_i}{\alpha_i-1} \qquad \forall\, n_i \geq 0,$$

As the length of the first job processed by a policy influences the completion time of all pending jobs, a straightforward calculation shows that

$$\mathbf{E}[\Pi_i^*(\boldsymbol{n},\boldsymbol{\omega},\boldsymbol{\alpha})] = \sum_{j=1}^{m} n_j\frac{\omega_i}{\alpha_i-1} + \int_0^\infty \mathbf{E}[\Pi^*(\boldsymbol{n}-\boldsymbol{e}_i,\boldsymbol{\omega}+x_i^1\boldsymbol{e}_i,\boldsymbol{\alpha}+\boldsymbol{e}_i)]f_i^0(x_i^1)\,\partial x_i^1, \tag{6.4}$$

for all $i \in [m]$ such that $n_i \geq 1$.

In the traditional stochastic scheduling variant, in which the parameters ϑ_i are known, the policy SEPT processes jobs in non-decreasing order of expected processing times. Since SEPT does not update jobs' expectations, SEPT processes in the Bayesian scheduling problem all jobs of a single job class en bloc, starting with the class having the shortest expected processing time. Formally, SEPT processes all jobs of class J_i before any job of class J_k if $\frac{\omega_i}{\alpha_i-1} < \frac{\omega_k}{\alpha_k-1}$. The random variable for the sum of completion times when SEPT is applied is denoted by Π^s, and its expected value can be written as

$$\mathbf{E}[\Pi^s(\boldsymbol{n},\boldsymbol{\omega},\boldsymbol{\alpha})] = \sum_{i=1}^{m}\frac{n_i(n_i+1)}{2}\frac{\omega_i}{\alpha_i-1} + \sum_{i=1}^{m-1}\sum_{j=i+1}^{m} n_in_j\min\left\{\frac{\omega_i}{\alpha_i-1},\frac{\omega_j}{\alpha_j-1}\right\}. \tag{6.5}$$

for all $\boldsymbol{n} \geq \mathbf{0}$, $\boldsymbol{\omega} > \mathbf{0}$ and $\boldsymbol{\alpha} > \mathbf{1}$. The non-adaptive character of SEPT could result in performance loss in comparison to a policy which makes use of additional information being revealed when processing the jobs. This shortcoming of SEPT is illustrated by the following example.

Example 6.2.1. Consider the Bayesian scheduling problem with two job classes. Let $\omega_1 = 10$, $\alpha_1 - 1 = 90$, $\omega_2 = 0.2$ and $\alpha_2 - 1 = 2$ such that $\mathbf{E}[X_1^1] = \frac{\omega_1}{\alpha_1-1} = \frac{10}{90} > 0.1$ and $\mathbf{E}[X_2^1] = \frac{\omega_2}{\alpha_2-1} = \frac{0.2}{2} = 0.1$, where X_i^1 denotes the processing time of the first job to be processed of class J_i. Since $\mathbf{E}[X_1^1] > \mathbf{E}[X_2^1]$, SEPT will first process all jobs of class J_2 and only afterwards all jobs of class J_1. However, the values are picked in such a way that the distribution of Θ_1 is peaked, i. e., the scheduler is relatively sure about the value of ϑ_1, whereas the distribution of Θ_2 is flat, i. e., the scheduler is relatively unsure about the value of ϑ_2 (see Figure 6.1). Consequently, it might be that actually $\vartheta_2 < \vartheta_1$, such that, in contrast to SEPT, it would be best to first start processing all jobs of class J_1. Similar to SEPT, OPT will start processing the jobs of class J_2 since $\mathbf{E}[X_2^1] < \mathbf{E}[X_1^1]$ and the beliefs regarding ϑ_2 are not that strong. However, in case $\vartheta_2 < \vartheta_1$, OPT will observe high processing times for the first few jobs of class J_2 and realize his mistake. After processing a few jobs of class J_2, OPT will therefore switch to processing jobs of class J_1, whereas SEPT continues with processing all jobs of class J_2. By choosing appropriate values for parameters $\boldsymbol{\omega}$ and $\boldsymbol{\alpha}$, the probability that $\vartheta_2 < \vartheta_1$ can be made even larger. Hence, the performance of SEPT can be far away from that of OPT. □

To overcome the shortcoming discussed in the example above, I propose an adaptive policy *learning*-SEPT (ℓ-SEPT). Whenever the machine is idle, this policy starts processing the job with shortest expected processing time. Thereby, it updates the expected processing time of

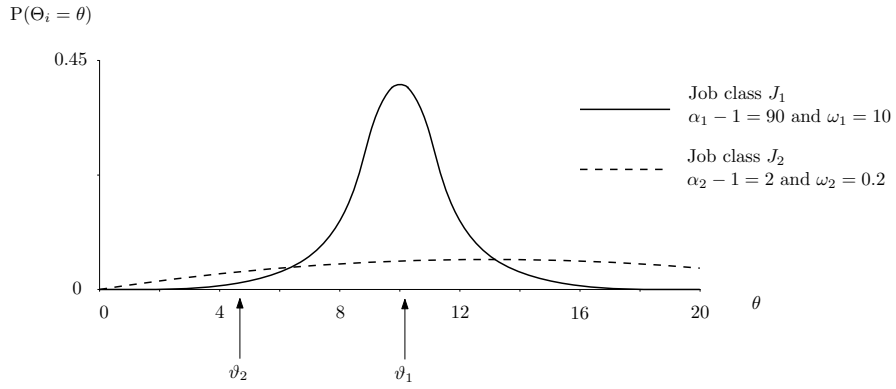

Figure 6.1: Two gamma distributions describing the beliefs with respect to the unknown parameters ϑ_1 and ϑ_2. The distribution corresponding to job class J_1 (J_2) is relatively peaked (flat) such that the scheduler is quite sure (unsure) about the value of ϑ_1 (ϑ_2).

jobs in a class every time a job of this specific class has been completed. Formally, after k_i jobs of each job class J_i have been finished, ℓ-SEPT starts processing a job of class J_h in case $h = \arg\min_i \left\{ \frac{\omega_i + \sum_{j=1}^{k_i} x_i^j}{\alpha_i + k_i - 1} \right\}$. Note that in Example 1, both SEPT and ℓ-SEPT start processing jobs of class J_2. However, in case $\vartheta_2 < \vartheta_1$, just like OPT, ℓ-SEPT will realize his mistake after having processed a few jobs of class J_2 and continue with processing jobs of class J_1. In what follows, Π^ℓ denotes the random variable for the sum of completion times when policy ℓ-SEPT is used.

To summarize, ℓ-SEPT uses more information than SEPT whereas OPT uses all available information, although none of the three policies know the values of ϑ_i. All three policies know the values of ω_i and α_i which are derived from the scheduler's beliefs about ϑ_i. From the value ω_i and α_i SEPT derives the first moment of the processing time of the first job of each class and bases its decision making on this knowledge. OPT and ℓ-SEPT are more intelligent in the sense that they make use of the underlying distribution of Θ_i and update this distribution in light of new realizations. OPT in particular uses $g_i(\theta|x_i^1, \ldots, x_i^k)$ through equations (6.1), (6.3), and (6.4). ℓ-SEPT actually only uses the first moment of the updated distribution of $(\Theta_i|X_i^1 = x_i^1, \ldots, X_i^k = x_i^k)$ to determine that the expected processing time of the next job of job class J_i equals (6.2), once k jobs of job class J_i have been processed. In terms of decision making, one could thus interpret OPT as having a long-term view whereas SEPT and ℓ-SEPT both have a short-term view. Both policies process a job of class J_i only if the expected processing time of the next job in this class is minimal. OPT, however, might choose to process a job of class J_i for which the expected processing time is not necessarily minimal. As a trade-off, OPT benefits from the additional information which is acquired regarding the uncertain parameter ϑ_i. This information could then lead to better future decision making and a lower sum of completion times.

6.3 Upper bound on performance guarantees

In this section, I prove that both SEPT and ℓ-SEPT have a performance guarantee at most m. First however, it is shown that the adaptive policy is indeed better than sequencing the jobs a priori.

Theorem 6.3.1. *For any $\boldsymbol{n} \geq \mathbf{0}$, $\boldsymbol{\omega} > \mathbf{0}$, and $\boldsymbol{\alpha} > \mathbf{1}$,*

$$\mathbf{E}[\Pi^{\ell}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] \leq \mathbf{E}[\Pi^{s}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})].$$

Proof. Define the function

$$z(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha}) := \sum_{i=1}^{m} \frac{n_i(n_i+1)}{2} \frac{\omega_i}{\alpha_i - 1} + \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} n_i n_j \min\left\{ \frac{\omega_i}{\alpha_i - 1}, \frac{\omega_j}{\alpha_j - 1} \right\}.$$

Then, by (6.5), the statement is equivalent to proving that

$$\mathbf{E}[\Pi^{\ell}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] \leq z(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha}), \tag{6.6}$$

which I will do by induction. For any $\boldsymbol{n} \leq \mathbf{1}$, (6.6) holds with equality (as there is no learning effect). Now, assume that (6.6) is true for any $\boldsymbol{n}' \leq \boldsymbol{n} - \boldsymbol{e}_i$ with any $i \in [m]$, and all $\boldsymbol{\omega} > \mathbf{0}$ and $\boldsymbol{\alpha} > \mathbf{1}$. Furthermore, assume that $h = \arg\min_{i \in [m]} \left\{ \frac{\omega_i}{\alpha_i - 1} \right\}$. Then, the sum of expected completion times of ℓ-SEPT can be bounded by

$$\begin{aligned}
&\mathbf{E}[\Pi^{\ell}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] = \sum_{i=1}^{m} n_i \frac{\omega_h}{\alpha_h - 1} + \int_0^{\infty} \mathbf{E}[\Pi^{\ell}(\boldsymbol{n} - \boldsymbol{e}_h, \boldsymbol{\omega} + x_h^1 \boldsymbol{e}_h, \boldsymbol{\alpha} + \boldsymbol{e}_h)] f_h^0(x_h^1)\, \partial x_h^1 \\
&\overset{\text{(induction)}}{\leq} \sum_{i=1}^{m} n_i \frac{\omega_h}{\alpha_h - 1} + \int_0^{\infty} z(\boldsymbol{n} - \boldsymbol{e}_h, \boldsymbol{\omega} + x_h^1 \boldsymbol{e}_h, \boldsymbol{\alpha} + \boldsymbol{e}_h)\, \partial x_h^1 \\
&\leq \sum_{i \neq h} n_i \frac{\omega_h}{\alpha_h - 1} + \sum_{i=1}^{m} \frac{n_i(n_i+1)}{2} \frac{\omega_i}{\alpha_i - 1} + \sum_{i \neq h} \sum_{j \geq i+1, j \neq h} n_i n_j \min\left\{ \frac{\omega_i}{\alpha_i - 1}, \frac{\omega_j}{\alpha_j - 1} \right\} \\
&\quad + \sum_{i \neq h} n_i (n_h - 1) \frac{\omega_h}{\alpha_h - 1} \\
&= \sum_{i=1}^{m} \frac{n_i(n_i+1)}{2} \frac{\omega_i}{\alpha_i - 1} + \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} n_i n_j \min\left\{ \frac{\omega_i}{\alpha_i - 1}, \frac{\omega_j}{\alpha_j - 1} \right\} = z(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha}),
\end{aligned}$$

where the last equality is due to the fact that $\frac{\omega_h}{\alpha_h - 1} = \min_{i \in [m]} \left\{ \frac{\omega_i}{\alpha_i - 1} \right\}$, by definition of h. □

Given the relation between SEPT and ℓ-SEPT, I next prove an upper bound on the performance guarantee of both SEPT and ℓ-SEPT. Hereby, I use a trivial lower bound on the performance of an arbitrary policy. This bound follows from the fact that in any policy jobs of a class have to wait for other jobs of the same class. Hence, neglecting waiting times caused by jobs having to wait for jobs of a different class the property follows.

Property 6.3.2. *Let Π be an arbitrary scheduling policy. Then, for any $\boldsymbol{n} \geq \mathbf{0}$, $\boldsymbol{\omega} > \mathbf{0}$, and $\boldsymbol{\alpha} > \mathbf{1}$,*

$$\mathbf{E}[\Pi(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] \geq \sum_{i=1}^{m} \mathbf{E}[\Pi(n_i \boldsymbol{e}_i, \boldsymbol{\omega}, \boldsymbol{\alpha})] = \sum_{i=1}^{m} \frac{(n_i+1)n_i}{2} \frac{\omega_i}{\alpha_i - 1}.$$

Theorem 6.3.3. *For any $\boldsymbol{n} \geq 0$, $\boldsymbol{\omega} > \mathbf{0}$, and $\boldsymbol{\alpha} > \mathbf{1}$,*

$$\frac{\mathbf{E}[\Pi^{\ell}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})]}{\mathbf{E}[\Pi^{*}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})]} \leq \frac{\mathbf{E}[\Pi^{s}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})]}{\mathbf{E}[\Pi^{*}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})]} \leq m.$$

Proof. The first inequality follows directly from Theorem 6.3.1. To prove the second inequality, it is assumed without loss of generality that $\omega_1/(\alpha_1 - 1) \leq \ldots \leq \omega_m/(\alpha_m - 1)$. Then,

$$\begin{aligned}
\mathbf{E}[\Pi^{s}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] &\overset{(6.5)}{=} \sum_{i=1}^{m} \frac{n_i(n_i+1)}{2} \frac{\omega_i}{\alpha_i - 1} + \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} n_i n_j \frac{\omega_i}{\alpha_i - 1} \\
&\overset{Lem.\,6.3.2}{\leq} \mathbf{E}[\Pi^{*}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] + \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} n_i n_j \frac{\omega_i}{\alpha_i - 1} \\
&\overset{\text{as } (n_i - n_j)^2 \geq 0}{\leq} \mathbf{E}[\Pi^{*}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] + \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \frac{1}{2}(n_i^2 + n_j^2) \frac{\omega_i}{\alpha_i - 1} \\
&\overset{\text{as } i<j}{\leq} \mathbf{E}[\Pi^{*}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] + \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \left(\frac{n_i(n_i+1)}{2} \frac{\omega_i}{\alpha_i - 1} + \frac{n_j(n_j+1)}{2} \frac{\omega_j}{\alpha_j - 1} \right) \\
&= \mathbf{E}[\Pi^{*}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] + \sum_{i=1}^{m} (m-1) \frac{n_i(n_i+1)}{2} \frac{\omega_i}{\alpha_i - 1} \overset{Lem.\,6.3.2}{\leq} m \cdot \mathbf{E}[\Pi^{*}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})].
\end{aligned}$$

□

6.4 Lower bounds on performance guarantees

The first part of this section shows that the bound given in Theorem 6.3.3 is tight for SEPT. The second part provides a lower bound of $1 + \sqrt{m-1}$ on the performance guarantee for ℓ-SEPT. An intuitive argument for this lower bound is provided in text. The rather technical proof follows in the appendix.

6.4.1 Tight performance guarantee of SEPT

I show that for any $\epsilon > 0$, there exists an instance for which the ratio of the value of SEPT to the value of OPT is only an additive ϵ away from the performance guarantee given in Theorem 6.3.3. In order to obtain this result, I first state some useful properties. The first two properties follow trivially from working out the integral. For the third property a proof will be presented.

Property 6.4.1. *For any $\omega > 0$ and $\alpha > 1$,*

$$\int_0^{\infty} \min\left\{ \frac{\omega_i + x_i^1}{\alpha_i}, 1 \right\} f_i^0(x_i^1)\, \partial x_i^1 = \frac{\omega_i}{\alpha_i - 1} - \frac{1}{\alpha_i - 1} \left(\frac{\omega_i}{\alpha_i} \right)^{\alpha_i}.$$

Property 6.4.2. *For any $\omega_i, \omega_j > 0$ and $\alpha_i, \alpha_j > 1$,*

$$\int_0^{\infty} \int_0^{\infty} \min\left\{ \frac{\omega_i + x_i^1}{\alpha_i}, \frac{\omega_j + x_j^1}{\alpha_j} \right\} f_i^0(x_i^1)\, \partial x_i^1 \cdot f_j^0(x_j^1)\, \partial x_j^1$$

$$= \frac{\omega_j}{\alpha_j - 1} - \frac{1}{\alpha_j - 1} \left(\frac{\alpha_i \omega_j}{\alpha_j \omega_i} \right)^{\alpha_j} \frac{\omega_i}{\alpha_j + \alpha_i - 1}.$$

Property 6.4.3. *For any* $\alpha > 1$*,*

$$\lim_{\alpha \downarrow 1} \frac{1}{\alpha - 1} \left(\frac{\alpha - 1}{\alpha} \right)^{\alpha} = 1$$

Proof. Let $\alpha > 1$. Then,

$$\begin{aligned}
\lim_{\alpha \downarrow 1} \frac{1}{\alpha - 1} \left(\frac{\alpha - 1}{\alpha} \right)^{\alpha} &= \lim_{\alpha \downarrow 1} \frac{1}{\alpha} \left(\frac{\alpha - 1}{\alpha} \right)^{\alpha - 1} = \lim_{\alpha \downarrow 1} \exp \left(\ln \left(\frac{1}{\alpha} \left(\frac{\alpha - 1}{\alpha} \right)^{\alpha - 1} \right) \right) \\
&= \lim_{\alpha \downarrow 1} \exp \left(\ln \left(\frac{(\alpha - 1)^{\alpha - 1}}{\alpha^{\alpha}} \right) \right) = \lim_{\alpha \downarrow 1} \exp \left(\ln \left((\alpha - 1)^{\alpha - 1} \right) - \ln \left(\alpha^{\alpha} \right) \right) \\
&= \lim_{\alpha \downarrow 1} \exp \left((\alpha - 1) \ln (\alpha - 1) - \ln \left(\alpha^{\alpha} \right) \right) = \exp (0 - 0) = 1.
\end{aligned}$$

□

Additionally, I derive a lower bound on SEPT and an upper bound on OPT.

Lemma 6.4.4. *For any* $\boldsymbol{n} \geq \mathbf{0}$*, there exist parameter settings* $\boldsymbol{\omega} > \mathbf{0}$*, and* $\boldsymbol{\alpha} > \mathbf{1}$ *such that* $\omega_1/(\alpha_1 - 1) < \omega_2/(\alpha_2 - 1) = \ldots = \omega_m/(\alpha_m - 1) = 1$*, and, for any* $\epsilon > 0$*,*

$$\mathbf{E}[\Pi^s(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] > \sum_{i=1}^{m} \frac{n_i(n_i + 1)}{2} + \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} n_i n_j - \epsilon.$$

Proof. For $\epsilon' > 0$ and arbitrary $\alpha_1 > 1$, let $\omega_1 = (1 - \epsilon')(\alpha_1 - 1)$ and $\omega_j = \alpha_j - 1$ for all $j \in \{2, \ldots, m\}$. By (6.5), it is that

$$\mathbf{E}[\Pi^s(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] = \frac{n_1(n_1 + 1)}{2}(1 - \epsilon') + \sum_{i=2}^{m} \frac{n_i(n_i + 1)}{2} + \sum_{j=2}^{m} n_1 n_j (1 - \epsilon') + \sum_{i=2}^{m-1} \sum_{j=i+1}^{m} n_i n_j,$$

for any $\boldsymbol{n} > \mathbf{0}$. Hence, for any $\epsilon > 0$, there exists an $\epsilon' > 0$ for which the lemma holds. □

Lemma 6.4.5. *For any* $\boldsymbol{n} \geq \mathbf{0}$*, there exist parameter settings* $\boldsymbol{\omega} > \mathbf{0}$*, and* $\boldsymbol{\alpha} > \mathbf{1}$ *such that* $\omega_1/(\alpha_1 - 1) < \omega_2/(\alpha_2 - 1) = \ldots = \omega_m(\alpha_m - 1) = 1$*, and, for any* $\epsilon > 0$*,*

$$\mathbf{E}[\Pi^*(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] < m \sum_{i=1}^{m} n_i + \sum_{i=1}^{m} \frac{n_i(n_i + 1)}{2} + \epsilon,$$

Proof. Consider the following policy Π: first process subsequently one job of class J_2 up to class J_m, observing realizations $x_2^1, \ldots, x_m^1$, and schedule all remaining jobs according to SEPT. Then,

$$\begin{aligned}
&\mathbf{E}[\Pi^*(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] \leq \mathbf{E}[\Pi(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] \\
&\quad = \sum_{i=1}^{m} n_i \sum_{j=2}^{m} \frac{\omega_j}{\alpha_j - 1} - \sum_{h=3}^{m} (h - 2) \frac{\omega_h}{\alpha_h - 1}
\end{aligned}$$

$$+\int_0^\infty \cdots \int_0^\infty \mathbf{E}[\Pi^s(\boldsymbol{n}-\sum_{i=2}^m \boldsymbol{e}_i, \boldsymbol{\omega}+\sum_{i=2}^m x_i^1 \boldsymbol{e}_i, \boldsymbol{\alpha}+\sum_{i=2}^m \boldsymbol{e}_i)]\, f_2^0(x_2^1)\, \partial x_2^1 \cdots f_m^0(x_m^1)\, \partial x_m^1.$$

As it is assumed that $\omega_i/(\alpha_i-1) \leq 1$ for all $i \in [m]$ it follows that

$$\sum_{i=1}^m n_i \sum_{j=2}^m \frac{\omega_j}{\alpha_j - 1} - \sum_{h=3}^m (h-2)\frac{\omega_h}{\alpha_h - 1} < \sum_{i=1}^m n_i \sum_{j=2}^m \frac{\omega_j}{\alpha_j - 1} \leq \sum_{i=1}^m n_i \sum_{j=2}^m 1 < m \sum_{i=1}^m n_i.$$

Therefore, it is left to prove that $\sum_{i=1}^m n_i(n_i+1)/2 + \epsilon$ will upper bound

$$\int_0^\infty \cdots \int_0^\infty \mathbf{E}[\Pi^s(\boldsymbol{n}-\sum_{i=2}^m \boldsymbol{e}_i, \boldsymbol{\omega}+\sum_{i=2}^m x_i^1 \boldsymbol{e}_i, \boldsymbol{\alpha}+\sum_{i=2}^m \boldsymbol{e}_i)]\, f_2^0(x_2^1)\, \partial x_2^1 \cdots f_m^0(x_m^1)\, \partial x_m^1.$$

Working out the integral and applying the definition of SEPT, see (6.5), it follows that the left hand side of the above inequality equals

$$\begin{aligned}
&\frac{n_1(n_1+1)}{2}\frac{\omega_1}{\alpha_1-1} + \sum_{i=2}^m \frac{(n_i-1)n_i}{2}\frac{\omega_i}{\alpha_i-1} \\
&+ \sum_{i=2}^m n_1(n_i-1)\int_0^\infty \min\left\{\frac{\omega_1}{\alpha_1-1}, \frac{\omega_i+x_i^1}{\alpha_i}\right\} f_i^0(x_i^1)\, \partial x_i^1 \\
&+ \sum_{i=2}^{m-1}\sum_{j=i+1}^m (n_i-1)(n_j-1)\int_0^\infty \int_0^\infty \min\left\{\frac{\omega_i+x_i^1}{\alpha_i}, \frac{\omega_j+x_j^1}{\alpha_j}\right\} f_i^0(x_i^1)\, \partial x_i^1\, f_j^0(x_j^1)\, \partial x_j^1,
\end{aligned}$$

which by Properties 6.4.1 and 6.4.2, and by assuming that $\omega_i/(\alpha_i-1) \leq 1$ for all $i \in [m]$, is upper bounded by

$$\begin{aligned}
&\frac{n_1(n_1+1)}{2} + \sum_{i=2}^m \frac{(n_i-1)n_i}{2} + \sum_{i=2}^m n_1(n_i-1)\left(\frac{\omega_i}{\alpha_i-1} - \frac{1}{\alpha_i-1}\left(\frac{\omega_i}{\alpha_i}\right)^{\alpha_i}\right) \\
&+ \sum_{i=2}^{m-1}\sum_{j=i+1}^m (n_i-1)(n_j-1)\left(\frac{\omega_j}{\alpha_j-1} - \frac{1}{\alpha_j-1}\left(\frac{\alpha_i\omega_j}{\alpha_j\omega_i}\right)^{\alpha_j}\frac{\omega_i}{\alpha_j+\alpha_i-1}\right).
\end{aligned}$$

Next, set $\omega_2 = \ldots = \omega_m = \alpha - 1$, $\alpha_2 = \ldots = \alpha_m = \alpha$ and let α tend to 1 from above. Then, the above quantity tends in its limit to the following:

$$\begin{aligned}
&\lim_{\alpha\downarrow 1}\left\{\frac{n_1(n_1+1)}{2} + \sum_{i=2}^m \frac{(n_i-1)n_i}{2} + \sum_{i=2}^m n_1(n_i-1)\left(1 - \frac{1}{\alpha-1}\left(\frac{\alpha-1}{\alpha}\right)^{\alpha}\right)\right. \\
&\qquad \left. + \sum_{i=2}^{m-1}\sum_{j=i+1}^m (n_i-1)(n_j-1)\left(1 - \frac{1}{2\alpha-1}\right)\right\} \\
&= \frac{n_1(n_1+1)}{2} + \sum_{i=2}^m \frac{(n_i-1)n_i}{2} + \lim_{\alpha\downarrow 1}\left\{\sum_{i=2}^{m-1}\sum_{j=i+1}^m (n_i-1)(n_j-1)\left(1 - \frac{1}{2\alpha-1}\right)\right\} \\
&= \frac{n_1(n_1+1)}{2} + \sum_{i=2}^m \frac{(n_i-1)n_i}{2},
\end{aligned}$$

where the first equality follows from Property 6.4.3. Hence, I conclude that for any $\boldsymbol{n} > \mathbf{0}$, $\omega_1 < \alpha_1 - 1$, there exists for any $\epsilon > 0$ an $\alpha^* > 1$ such that for all $1 < \alpha_2 = \ldots = \alpha_m = \omega_2 + 1 = \ldots = \omega_m + 1 < \alpha^*$, and

$$\mathbf{E}[\Pi^*(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] < m \sum_{i=1}^{m} n_i + \sum_{i=1}^{m} \frac{(n_i + 1)n_i}{2} + \epsilon.$$

□

Lemmata 6.4.4 and 6.4.5, and letting the number of jobs in each class tend to infinity with the same rate, then yield the following theorem:

Theorem 6.4.6. *There exist parameter settings $\boldsymbol{n} > \mathbf{0}$, $\boldsymbol{\omega} > \mathbf{0}$ and $\boldsymbol{\alpha} > \mathbf{1}$, such that*

$$\frac{\mathbf{E}[\Pi^s(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})]}{\mathbf{E}[\Pi^*(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})]} > m - \epsilon,$$

for any $\epsilon > 0$.

Proof. The restrictions imposed on the values $\boldsymbol{\alpha}$ and $\boldsymbol{\omega}$ in Lemmas 6.4.4 and 6.4.5 can be satisfied simultaneously. Further, set $n_1 = \ldots = n_m = n$ and let n tend to infinity. Then,

$$\begin{aligned}\lim_{n\to\infty} \frac{\mathbf{E}[\Pi^s(n, \ldots, n, \boldsymbol{\omega}, \boldsymbol{\alpha})]}{\mathbf{E}[\Pi^*(n, \ldots, n, \boldsymbol{\omega}, \boldsymbol{\alpha})]} &> \lim_{n\to\infty} \frac{mn^2 + mn + m(m-1)n^2}{2m^2 n + mn^2 + mn} - \epsilon \\ &= \lim_{n\to\infty} \frac{mn + 1}{n + 1 + 2m} - \epsilon = m - \epsilon.\end{aligned}$$

□

6.4.2 Lower bound on the performance guarantee of ℓ-SEPT

In the current subsection it is shown that the performance guarantee of ℓ-SEPT is no less than $1 + \sqrt{m-1}$. Here, I will provide an intuitive reasoning whereas for a full proof I refer the reader to Appendix 6.A.

The underlying idea is to create a bad instance for ℓ-SEPT which is similar in spirit as the worst case instance of SEPT which was discussed in the previous subsection in proof of Lemma 6.4.4. In that bad SEPT instance, the parameters were set in such a way that $\mathbf{E}[X_1^1]$ slightly undercuts $\mathbf{E}[X_i^1]$, for all $i \in \{2, \ldots, m\}$. Hence, SEPT starts processing all jobs of class J_1, only followed by the jobs of the other classes afterwards. OPT however, starts processing jobs of the other classes $i \neq 1$ as the distribution of the corresponding Θ_i's has been chosen to be extremely flat. Therefore, it is beneficial to process a few jobs of the classes $i \neq 1$ to find out whether the value of ϑ_i actually exceeds the value of ϑ_1 resulting in lower processing requirements for jobs of job class J_i. To create a bad instance for ℓ-SEPT, I want to preserve this bad structure. Therefore, it is needed that ℓ-SEPT does not switch to processing jobs from a second class after it processed a few jobs of job class J_1. By setting the values of ω_1 and α_1 extremely large, the value of ϑ_1 is highly certain. Consequently, the realizations of processing times of jobs from class J_1 barely affect the expected processing time for the next job of job

class J_1 to be processed, i. e., when ω_1 and α_1 are big enough it is that, after k observations on the first job class,

$$\frac{\omega_1 + \sum_{j=1}^{k} x_1^j}{\alpha_1 + k - 1} \approx \frac{\omega_1}{\alpha_1 - 1} = 1 - \epsilon < 1 = \frac{\omega_2}{\alpha_2 - 1} = \ldots = \frac{\omega_m}{\alpha_m - 1}.$$

The theorem follows from adhering this instance structure and working out the technical details. A full proof is given in Appendix 6.A.

Theorem 6.4.7. *There exist parameter settings $\boldsymbol{n} \geq \mathbf{0}$, $\boldsymbol{\omega} > \mathbf{0}$, $\boldsymbol{\alpha} > \mathbf{1}$ such that, for any $\epsilon > 0$,*

$$\frac{\mathbf{E}[\Pi^{\ell}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})]}{\mathbf{E}[\Pi^{*}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})]} > 1 + \sqrt{m-1} - \epsilon.$$

6.5 Tighter performance bound for ℓ-SEPT

In this section, I conjecture that the performance guarantee of ℓ-SEPT with respect to OPT is upper bounded by $1 + \sqrt{m-1}$, thereby closing the gap between upper and lower bound of Theorems 6.3.3 and 6.4.7. In order to motivate this conjecture, I first derive a lower bound on OPT and an upper bound on the performance of ℓ-SEPT. The aggregation of these two lemmata together with some intuitive reasoning leads to the proposed conjecture. The second subsection supports the conjecture by presenting some empirical evidence.

6.5.1 A conjectured performance guarantee for ℓ-SEPT

For the scheduling problem in which all parameter values are known, SEPT is an optimal policy. The minimum total completion time in expectation under complete information for parameters $\theta_1, \ldots, \theta_m$, is thus given by

$$\sum_{i=1}^{m} \frac{n_i(n_i+1)}{2} \frac{1}{\theta_i} + \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} n_i n_j \min\left\{\frac{1}{\theta_i}, \frac{1}{\theta_j}\right\}. \tag{6.7}$$

In case each parameter is a random variable Θ_i instead, the expected value of (6.7) with respect to the prior parameter distribution $g_i(\theta)$ gives a lower bound on the total completion time in expectation of an arbitrary policy Π, see also Burnetas and Katehakis [26]. That is,

$$\mathbf{E}[\Pi(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] \geq \sum_{i=1}^{m} \frac{n_i(n_i+1)}{2} \frac{\omega_i}{\alpha_i - 1} + \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} n_i n_j \, \mathbf{E}\left[\min\left\{\frac{1}{\Theta_i}, \frac{1}{\Theta_j}\right\}\right], \tag{6.8}$$

for any $\boldsymbol{n} \geq \mathbf{0}$, $\boldsymbol{\omega} > \mathbf{0}$, and $\boldsymbol{\alpha} > \mathbf{1}$. Hereby, note that $\mathbf{E}[\frac{1}{\Theta_i}] = \frac{\omega_i}{\alpha_i - 1}$ for all $i \in [m]$.

The expectations on the right hand side of (6.8) can be rewritten using the following insight. Let the random variable $Y := \min\left\{\frac{1}{\Theta_i}, \frac{1}{\Theta_j}\right\}$ for some $i \neq j$, and let $F_Y(\cdot)$ be the corresponding cumulative distribution function. Moreover, let $F_{\Theta_i^{-1}}(\cdot)$ denote the cumulative distribution function of random variable Θ_i^{-1}, the processing time of jobs in class i. Then,

$$1 - F_Y(y) = 1 - \mathbf{Pr}[Y \leq y] = \mathbf{Pr}\left[\frac{1}{\Theta_i} > y\right] \mathbf{Pr}\left[\frac{1}{\Theta_j} > y\right] = \left(1 - F_{\Theta_i^{-1}}(y)\right)\left(1 - F_{\Theta_j^{-1}}(y)\right)$$

$$= \left(1 - F_{\Theta_i^{-1}}(y)\right) + \left(1 - F_{\Theta_j^{-1}}(y)\right) - \left(1 - F_{\Theta_i^{-1}}(y) F_{\Theta_j^{-1}}(y)\right)$$

such that

$$\mathbf{E}\left[\min\left\{\frac{1}{\Theta_i}, \frac{1}{\Theta_j}\right\}\right] = \int_0^\infty (1 - F_Y(y))\, \partial y = \frac{\omega_i}{\alpha_i - 1} + \frac{\omega_j}{\alpha_j - 1} - \int_0^\infty \left(1 - F_{\Theta_i^{-1}}(y) F_{\Theta_j^{-1}}(y)\right) \partial y.$$

Working out the last integral, plugging it into (6.8), and rewriting leads to the following result:

Lemma 6.5.1. *For any $\boldsymbol{n} \geq \mathbf{0}$, $\boldsymbol{\omega} > \mathbf{0}$ and $\boldsymbol{\alpha} > \mathbf{1}$ with $\frac{\omega_1}{\alpha_1 - 1} \leq \frac{\omega_2}{\alpha_2 - 1} \leq \ldots \leq \frac{\omega_m}{\alpha_m - 1}$,*

$$\mathbf{E}[\Pi^*(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] \geq \sum_{i=1}^m \frac{n_i(n_i+1)}{2} \frac{\omega_i}{\alpha_i - 1} + \sum_{i=1}^{m-1} \sum_{j=i+1}^m n_i n_j\, \xi(i,j)\, \frac{\omega_i}{\alpha_i - 1},\ \textit{where}$$

$$\xi(i,j) = 1 - \frac{\left(\frac{\omega_i}{\omega_i + \omega_j}\right)^{\alpha_i - 1} \left(\frac{\omega_j}{\omega_i + \omega_j}\right)^{\alpha_j - 1}}{(\alpha_i + \alpha_j - 1) \cdot B(\alpha_i, \alpha_j)} \in (0,1)\,,$$

and $B(\alpha_i, \alpha_j) = \int_0^1 t^{\alpha_i - 1}(1-t)^{\alpha_j - 1}\, \partial t$ denotes a Beta function.

The proof of this lemma and of all remaining results in this subsection are presented in Appendix 6.C.

In order to derive a stronger upper bound than (6.6) on the performance of ℓ-SEPT, first observe that the performance of ℓ-SEPT given an arbitrary amount of job classes can be expressed as a linear combination of the performances of ℓ-SEPT given one and two job classes.

Lemma 6.5.2. *For any $\boldsymbol{n} \geq \mathbf{0}, \boldsymbol{\omega} > \mathbf{0}$ and $\boldsymbol{\alpha} > \mathbf{1}$,*

$$\mathbf{E}[\Pi^\ell(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] = \sum_{i=1}^{m-1} \sum_{j=i+1}^m \mathbf{E}[\Pi^\ell(n_i \boldsymbol{e}_i + n_j \boldsymbol{e}_j, \boldsymbol{\omega}, \boldsymbol{\alpha})] - (m-2) \sum_{i=1}^m \mathbf{E}[\Pi^\ell(n_i \boldsymbol{e}_i, \boldsymbol{\omega}, \boldsymbol{\alpha})].$$

Based on this lemma and assuming without loss of generality that $\frac{\omega_1}{\alpha_1 - 1} \leq \frac{\omega_2}{\alpha_2 - 1} \leq \ldots \leq \frac{\omega_m}{\alpha_m - 1}$, the following upper bound on the performance of ℓ-SEPT is derived:

Lemma 6.5.3. *For any $\boldsymbol{n} \geq \mathbf{0}, \boldsymbol{\omega} > \mathbf{0}$ and $\boldsymbol{\alpha} > \mathbf{1}$ with $\frac{\omega_1}{\alpha_1 - 1} \leq \frac{\omega_2}{\alpha_2 - 1} \leq \ldots \leq \frac{\omega_m}{\alpha_m - 1}$,*

$$\mathbf{E}[\Pi^\ell(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] \leq \sum_{i=1}^m \frac{n_i(n_i+1)}{2} \frac{\omega_i}{\alpha_i - 1} + \sum_{i=1}^{m-1} \sum_{j=i+1}^m \left((n_i - 1) n_j \frac{\omega_i}{\alpha_i - 1}\, \zeta(i,j) + n_j \frac{\omega_i}{\alpha_i - 1}\right),$$

$$\textit{where } \zeta(i,j) = \left(1 - \frac{1}{\alpha_i}\left(\frac{\omega_i(\alpha_j - 1)}{\alpha_i \omega_j}\right)^{\alpha_i - 1}\right) \in (0,1)\,.$$

The proof of this lemma first applies Lemma 6.5.2 to the performance of ℓ-SEPT given m job classes. Then, it uses the assumed order on the expected processing times to schedule exactly one (minimum expected processing time) job for each ℓ-SEPT given two job classes. Finally, (6.6) is applied to upper bound each expected sum of completion times of ℓ-SEPT with two job classes, and all integrals in the obtained expression are replaced by the following generalized version of Property 6.4.1:

Property 6.5.4. *For any* $\omega_i, \omega_j > 0$ *and* $\alpha_i, \alpha_j > 1$*,*

$$\int_0^\infty \min\left\{\frac{\omega_i + x_i^1}{\alpha_i}, \frac{\omega_j}{\alpha_j - 1}\right\} f_i^1(x_i^1) dx_i^1 = \frac{\omega_i}{\alpha_i - 1}\left[1 - \frac{1}{\alpha_i}\left(\frac{\omega_i(\alpha_j - 1)}{\alpha_i \omega_j}\right)^{\alpha_i - 1}\right].$$

Up to now, each step in this subsection has been proven. From here on though, I will proceed by intuition to reach the conjecture as I was not able to simplify the ratio of the quantities given in Lemmas 6.5.1 and 6.5.3 further.

From Lemmas 6.5.1 and 6.5.3, it follows that both the lower bound on OPT and the upper bound on ℓ-SEPT contain the term $\sum_{i=1}^m \frac{n_i(n_i+1)}{2}\frac{\omega_i}{\alpha_i - 1}$. In order to find the worst case ratio of $\mathbf{E}[\Pi^\ell(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})]$ over $\mathbf{E}[\Pi^*(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})]$, I therefore need $\xi(i,j)$ and $\zeta(i,j)$ to tend to 0 and 1, respectively, for almost all combinations of $i \neq j$. This resembles the way in which the lower bound for SEPT in Section 6.4.1 was constructed. For $\xi(i,j)$ to tend to 0, either (i) $\alpha_i = \omega_i + 1$ holds and this quantity tends to 1; or (ii) $\alpha_j = \omega_j + 1$ holds and this quantity tends to 1. However, for $\zeta(i,j)$ to tend to 1, it is needed that either α_i or α_j tends to infinity. In the Bayesian scheduling problem with two job classes, J_1 and J_2, one can simultaneously establish that $\xi(1,2)$ tends to 0 and $\zeta(1,2)$ tends to 1 by letting α_1 approach infinity while letting $\omega_2 = \alpha_2 - 1$ tend to 0. Confronted with a third class, J_3, the only way to bring $\xi(2,3)$ to 0 and $\zeta(2,3)$ to 1 is by letting α_3 approach infinity as well. This will, however, cause $\xi(1,3)$ to tend to 1, thereby increasing the denominator of the ratio, which is undesirable. Therefore, it seems that the best thing one can do is to set $\alpha_j = \omega_j + 1$, while letting ω_j tend to 0, for all classes $j \geq 2$. Within this structure $\xi(i,j) = 0$ for all $i \neq j$, $\zeta(1,j) = 1$ for all $j > 1$, and $\zeta(i,j) = 0$ for all $i, j > 1$.

By choosing n_1 bigger than n_j for all $j \geq 2$, some extra weight is given to $\zeta(1,j)$ in the numerator. Increasing the number of jobs in class J_1, however, also increases the term $\frac{n_1(n_1+1)}{2}$ in the denominator, thereby decreasing the ratio again. Due to this trade-off, the optimal choice is $n_1 = n\sqrt{m-1}$ and $n_j = n$ for all $j \geq 2$, with n going to infinity. Note that this structure leads to the derivation of the lower bound for ℓ-SEPT as in Subsection 6.4.2. The preceding observations bring us to the following conjecture.

Conjecture 6.5.5. *For any* $\boldsymbol{n} \geq \mathbf{0}, \boldsymbol{\omega} > \mathbf{0}$ *and* $\boldsymbol{\alpha} > \mathbf{1}$*,*

$$\frac{\mathbf{E}[\Pi^\ell(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})]}{\mathbf{E}[\Pi^*(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})]} \leq 1 + \sqrt{m-1}.$$

I remark that in my motivation for the conjecture only extreme values for the job class parameters were considered. That is, either $\alpha_i = \omega_i + 1$ is chosen to tend to 1 or to infinity. Reason for this is that choosing α_i somewhere in the middle of its range seems not to be beneficial. In that case, $\xi(i,j)$ will be either significantly larger than 0 or significantly smaller than 1, depending on how the parameters of job class $j \neq i$ are chosen. In the next section, I provide some computational results on three job classes to support this claim, and also the conjecture in general.

Moreover, note that the above conjecture together with Theorem 6.4.7 would yield a tight analysis of the performance guarantee for ℓ-SEPT.

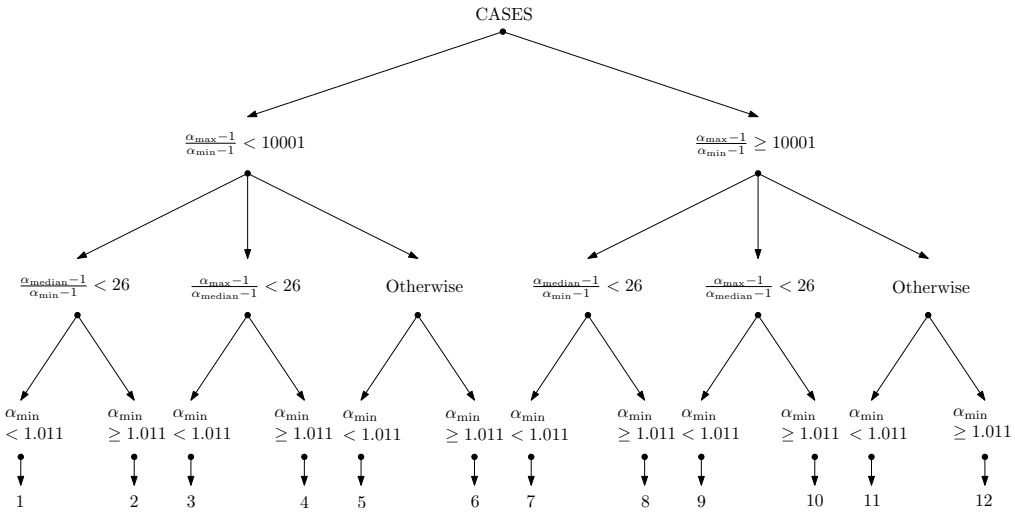

Figure 6.2: The 12 different case settings considered.

6.5.2 Empirical evidence supporting the conjecture

The lower bound in Lemma 6.5.1 and upper bound in Lemma 6.5.3 can be used to investigate the performance of ℓ-SEPT with respect to OPT. This is done by calculating the ratio of these bounds for several Bayesian scheduling instances. Initially, I consider only 3 job classes. The instances studied are as follows: the gamma prior settings are set such that $\omega_i, (\alpha_i - 1) \in \{0.0001, 0.0002, 0.0005, \ldots, 100, 200, 500\}$ for each job class J_i ($i \in \{1, 2, 3\}$). For each of these settings, I further calculate the worst possible ratio over the following numbers of jobs in each class: $n_1, n_2, n_3 \in \{10, 20, 50, 100, 200, 400, \ldots, 1800, 2000\}$. This results in almost 240 billion different computations, covering a large amount of the possible parameter settings. All these computations are performed in MATLAB.

Preliminary computations show that ℓ-SEPT performs relatively good when the a priori expected processing times of the job classes differ significantly, i. e., when

$$\max_{i\in[m]} \left\{ \frac{\omega_i}{\alpha_i - 1} \right\} > 2 \cdot \min_{j\in[m]} \left\{ \frac{\omega_j}{\alpha_j - 1} \right\}. \tag{6.9}$$

Intuitively, learning will only be beneficial in this case with a very small probability such that ℓ-SEPT and OPT will process the job classes in the same order. Therefore, I focus on the parameter settings with $\boldsymbol{\omega}$ and $\boldsymbol{\alpha}$ opposing (6.9). These are presented in Figure 6.2, where

$$\alpha_{\max} = \max_{i\in\{1,2,3\}} \alpha_i; \qquad \alpha_{\text{median}} = \operatorname{median}_{i\in\{1,2,3\}} \alpha_i; \qquad \text{and } \alpha_{\min} = \min_{i\in\{1,2,3\}} \alpha_i.$$

As can be seen in the figure, I distinguish between 12 different cases, depending on the following points:

Case	$\rho_{\max}$	ω_1	ω_2	ω_3	α_1	α_2	α_3	$\frac{n_1}{100}$	$\frac{n_2}{100}$	$\frac{n_3}{100}$	$\overline{\rho}$
01	2.34	5	0.0005	0.0005	11	1.001	1.001	20	14	14	1.21
02	2.06	0.2	0.0001	0.0001	51	1.02	1.02	16	10	10	1.26
03	2.10	5	0.2	0.0005	6	1.2	1.0005	18	10	18	1.61
04	1.87	2	0.2	0.005	11	2	1.02	12	8	14	1.36
05	2.17	1	0.0001	0.005	6	1.0005	1.02	18	14	10	1.77
06	1.87	1	0.02	0.0005	51	2	1.02	12	8	14	1.55
07	2.41	500	0.0001	0.0001	501	1.0001	1.0001	20	14	14	1.79
08	2.06	100	0.005	0.005	501	1.02	1.02	16	10	10	1.51
09	2.11	50	2	0.001	6	1.2	1.0001	18	10	18	1.87
10	1.79	100	10	0.005	501	51	1.02	8	8	14	1.69
11	2.38	500	0.005	0.0001	501	1.005	1.0001	14	10	10	1.87
12	1.87	100	0.2	0.005	501	2	1.02	12	8	14	1.69

Table 6.1: For 12 different cases the largest and average ratio of the bounds on $\mathbf{E}[\Pi^{\ell}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})]$ and $\mathbf{E}[\Pi^{*}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})]$ as given by Lemmata 6.5.1 and 6.5.3.

- $\alpha_{\min}$ and $\alpha_{\max}$ are close, or not;
- α_{median} is close to $\alpha_{\min}$, close to $\alpha_{\max}$, or somewhere in between;
- $\alpha_{\min}$ is close to its lower limit (which equals one).

The results are presented in Table 6.1. For each case, the largest ratio encountered is reported along with the corresponding parameters, and the average ratio. The largest ratio encountered is 2.4097 (Case 7), which is close to the lower bound of $1 + \sqrt{m-1} = 1 + \sqrt{2} \approx 2.4142$. This ratio is obtained for the setting with $\frac{\omega_i}{\alpha_i - 1} = 1$ for all i, two job classes being very uncertain (α_2 and α_3 close to one), and one job class very certain (α_1 taking the highest value allowed in our settings). Moreover, the corresponding values for n_1, n_2 and n_3 are as large as possible, while preserving that the number of jobs in the class with certain jobs, n_1, is roughly a factor of $\sqrt{m-1} = \sqrt{2}$ larger than the number of jobs in the other job classes, n_2 and n_3.

Overall, the structure of the instance yielding the highest ratio seems to adhere the exact same structure as in the construction of the lower bound on the performance ratio of ℓ-SEPT in Subsection 6.4.2. Therefore, the computations support Conjecture 6.5.5. Moreover, Cases 1 and 11, which also return a large ratio, mirror this structure as well but within the limitations of their case setting. It therefore comes as no surprise that these cases also yield high performance ratios.

Considering the average performance ratios per case, it follows that the average ratio is positively correlated with the ratio $\alpha_{\max}/\alpha_{\min}$. Given that $\alpha_{\min}$ is close to one and having a high value for $\alpha_{\max}$, it does not seem to matter that much where α_{median} is (compare Cases 7, 9 and 11). To find the worst case instance though, it matters since having a α_{median} far away from $\alpha_{\min}$ only gives a performance ratio of at most 2.3784 (see Cases 9 and 11), which supports my claim at the end of the previous subsection that all non-biggest α should be close to $\alpha_{\min}$ to reach the worst possible performance ratio.

So far, the ratio of the upper bound in Lemma 6.5.1 over the lower bound in Lemma 6.5.3 has only been considered for 3 job classes. To conclude this subsection, I will have a brief look at the behavior of this ratio in case there are more than 3 job classes. I focus on the case

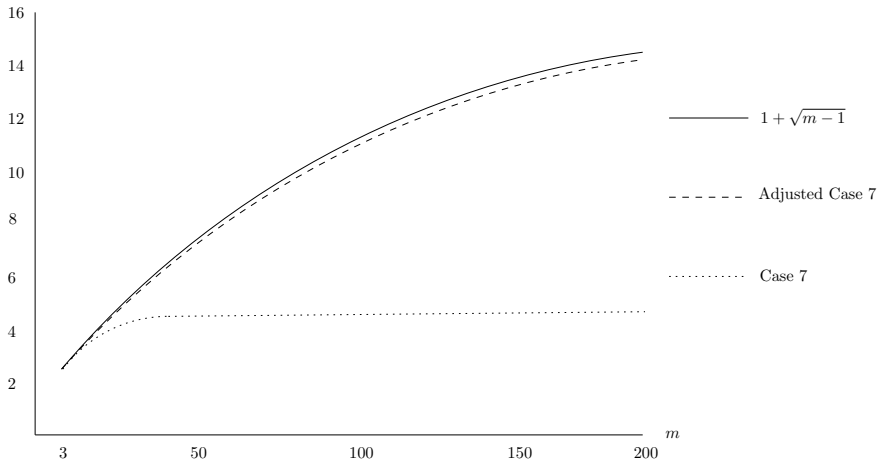

Figure 6.3: Largest ratio for the setting of Case 7 with more than 3 job classes

setting that previously, in the presence of 3 job classes, gave the largest ratio, that is Case 7:

$$\alpha_1 - 1 = \omega_1 = 500, \qquad \alpha_2 - 1 = \omega_2 = \ldots = \alpha_m - 1 = \alpha_m = 0.0001,$$
$$n_1 = 2000, \qquad n_2 = \ldots = n_m = 1400,$$

and an adjusted version of this case:

$$\alpha_1 - 1 = \omega_1 = 10000, \qquad \alpha_2 - 1 = \omega_2 = \ldots = \alpha_m - 1 = \alpha_m = 0.0001,$$
$$n_1 = \sqrt{m-1} \cdot 10000, \qquad n_2 = \ldots = n_m = 10000,$$

and compute the ratios for $m \in \{3, \ldots, 200\}$. The results are given in Figure 6.3.

The figure shows that the computed ratios for the Case 7 setting are very close to $1 + \sqrt{m-1}$ for small number of job classes m. As soon as m increases, a more extreme setting is necessary to still get close to $1 + \sqrt{m-1}$. This setting is provided by the adjusted version of Case 7, which is in agreement with the construction of the lower bound for SEPT in Subsection 6.4.2 and the intuition in Subsection 6.5.1.

In summary, although I performed over 240 billion computations, I was not able to find instances which contradict Conjecture 6.5.5. Moreover, Table 6.1 hinted that the structure used to construct the lower bound on ℓ-SEPT in Subsection 6.4.2 actually is the worst case instance structure. If that is true, then Lemmata 6.5.1 and Lemma 6.5.3 imply that the ratio of $\mathbf{E}[\Pi^{\ell}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})]$ over $\mathbf{E}[\Pi^{*}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})]$ is no more than $1 + \sqrt{m-1}$ since the numerical values of this ratio can approach the value $1 + \sqrt{m-1}$ arbitrarily close for this instance structure, but not surpass it. Hence, the empirics support Conjecture 6.5.5.

6.6 Empirical Analysis

In this section, I investigate the performance of SEPT and ℓ-SEPT with respect to the optimal value in a Bayesian setting. As the optimal dynamic program is highly impractical due to its computation time and failure to to achieve a high enough accuracy to deal with all possible instances, simple but powerful methods are needed to come up with good solutions fast. SEPT and ℓ-SEPT pose two such policies. Here, I compare the two policies and analyze their behavior with respect to the optimal policy. Especially ℓ-SEPT proves to be a powerful tool which often returns a solution whose quality is close to the quality of the optimal solution. All computations in this section are performed using MATLAB.

Per instance, I perform a large amount of simulations to compare the average values of SEPT and ℓ-SEPT with the average optimal Bayesian solution. In order to compute the values of OPT, I used the dynamic programming algorithm presented in Section 4 of the paper of Hamada and Glazebrook [63]. Note that in contrast to the previous subsection, I no longer calculate upper bounds on the expected performance ratio but compare empirical average performance ratios instead. As a consequence though, the set of different settings which I could test for is limited by the practicality of the optimal dynamic program which fails for large instances and when the values for α and ω either approach their lower limits or have a too high accuracy.

The Bayesian scheduling instances studied are as follows: I consider 3 different problem settings; that is, there are either 2, 3 or 5 different job classes. To study how large the gap between SEPT and ℓ-SEPT might be, I set the number of jobs in each class equal to 15. Having an equal number of jobs in each class, maximizes the value of SEPT with respect to OPT (though this is not true for LSEPT). Further, the gamma prior parameters are set such that ω_i and $(\alpha_i - 1)$ are both an element of $\{0.5\,;\, 1.0\,;\, 5.0\,;\, 25.0\}$, for each job class J_i, $i \in \{1, \ldots, 5\}$. This results in 100, 400, and 3136 different computations when there are 2, 3 or 5 job classes, respectively, thereby covering the majority of interesting instances, i. e., the cases in which all job classes have either high or low parameter uncertainty, and the mixed case in which some class has high and some other one has low parameter uncertainty. Moreover, these computations could still be performed in a reasonable amount of time. Choosing our settings in a more extreme fashion (like having more jobs in each job class or lower values for ω_i and α_i) immediately results in difficulties with the precision in calculating OPT, and also significantly increases the computation time of this optimal policy. On average, it took about 4 minutes to calculate one setting through. The resulting total computation time was more than 9 days to evaluate all chosen settings.

In my computations, 50.000 simulations are run to compute the average values of SEPT, ℓ-SEPT and OPT for each Bayesian scheduling instance. In each of those simulations, I draw for each job class a parameter realization from a gamma distribution. This realization is subsequently used to draw 15 processing requirement realizations from an exponential distribution. Using these realizations the sum of completion times for each of the policies is calculated. As mentioned before, performance of the policies SEPT and ℓ-SEPT is measured by average objective value of the policy over the average objective value of OPT.

The computational results are presented in Appendix 6.C, Tables 6.2, 6.3, and 6.4. A first observation shows that ℓ-SEPT significantly outperforms SEPT in every setting, regardless whether there are 2, 3 or 5 job classes. When the value of SEPT is far away from OPT, the

deviation from ℓ-SEPT to OPT is at least a factor 3 less than the deviation from SEPT to OPT. On the other hand, when the values of SEPT and OPT are close, then ℓ-SEPT usually equals OPT. Moreover, the computational results in Tables 6.2 and 6.3 indicate that whenever more than one job class has a high value for α_i, say $\alpha_i \geq 6$, SEPT and ℓ-SEPT both perform reasonably well: SEPT is typically at most 10% away from OPT whereas ℓ-SEPT is at most 2% away. In case of 5 job classes I therefore only present that part of the table, i. e., I suppressed many settings for which multiple classes J_i have a high value for α_i.

The performance of ℓ-SEPT is poorest when one job class, say J_i, has a low parameter uncertainty ($\alpha_i - 1 = \omega_i$ and both values large) and all other classes, J_j with $j \neq i$, have high parameter uncertainty ($\alpha_j - 1 = \omega_j$ and both values low). To be more explicit, the worst case for ℓ-SEPT is obtained whenever $\alpha_1 - 1 = \omega_1 = 25$ and $\alpha_j - 1 = \omega_j = 0.5$ for all other job classes $j \neq 1$: we find that ℓ-SEPT is no more than 9.1%, 10.5% and 12.5% away from OPT with 2, 3 or 5 job classes, respectively. Although ℓ-SEPT performs worse when more job classes are present, the rate by which gap between ℓ-SEPT and OPT is increasing is relatively low. It is further interesting to observe that instances for which ℓ-SEPT performs relatively poor adhere the structure used to construct the lower bound in Subsection 6.4.2. Consequently, the computational results again support the there stated conjecture.

The performance of SEPT significantly deteriorates in the presence of more job classes. In the worst setting, I find that SEPT is up to 32.6%, 51.0% and 69.8% away from OPT when having 2, 3 or 5 job classes, respectively. Although SEPT also performs poorly for the instances where ℓ-SEPT performs relatively bad, the worst case instances are those where $\alpha_i = 1.5$ and $\omega_i = 1$ for all job classes J_i.

In order for both ℓ-SEPT and SEPT to perform bad, we need that all job classes have high parameter uncertainty and almost equal expected processing times of the jobs for the different classes. In that case, the risk is the highest that the two policies start processing the 'wrong' job class. This is indeed the instance for which SEPT performs worst in our computational results. However, since ℓ-SEPT updates the expected processing time of a class whenever a job of that class has been completed, it should be made sure that ℓ-SEPT does not switch job classes once it started processing the 'wrong' one. Therefore, ℓ-SEPT needs this job class to have very low parameter uncertainty, which is again confirmed by my computational results.

6.7 Concluding remarks

This chapter studied the performance guarantee of SEPT and its natural extension ℓ-SEPT on a Bayesian scheduling problem with m different job classes. The dynamic programming algorithm formulated by Hamada and Glazebrook [63] for the studied problem is optimal but computationally expensive and limited. This calls for the need to develop policies of low computational effort which yield good qualitative performance. The policies SEPT and its adaptive variant ℓ-SEPT are examples of such policies and I showed upper and lower bounds on the performance guarantee of both policies. In the previous section, I compared the empirical performance of these two policies with each other and with respect to an optimal policy, OPT. On all instances ℓ-SEPT clearly outperforms the non-adaptive variant SEPT, thereby emphasizing the impact of learning on the performance of the algorithm. Especially, ℓ-SEPT has shown to be a powerful policy which is almost optimal on many instances. Additionally, I

remark ℓ-SEPT is more robust than SEPT, that is, SEPT has a much higher variance in the value of the returned solution than the other two policies do. This feature is explained by the fact that SEPT does not revoke its decision once made a bad choice.

In a precursor of this work I showed together with my co-authors an upper bound performance guarantee for SEPT and ℓ-SEPT in case of having just 2 job classes:

Theorem 6.7.1. *For any* $n_1 \geq 1$, $n_2 \geq 1$, $\omega_1 > 0$, $\omega_2 > 0$, $\alpha_1 > 1$ *and* $\alpha_2 > 1$*, it holds that*

$$\frac{\mathbf{E}[\Pi^{\ell}(n_1, n_2, \omega_1, \omega_2, \alpha_1, \alpha_2)]}{\mathbf{E}[\Pi^{*}(n_1, n_2, \omega_1, \omega_2, \alpha_1, \alpha_2)]} \leq \frac{\mathbf{E}[\Pi^{s}(n_1, n_2, \omega_1, \omega_2, \alpha_1, \alpha_2)]}{\mathbf{E}[\Pi^{*}(n_1, n_2, \omega_1, \omega_2, \alpha_1, \alpha_2)]} \leq \frac{n_1^2 + n_2^2 + 2n_1n_2 + n_1 + n_2}{n_1^2 + n_2^2 + n_1 + n_2 + 2\min\{n_1, n_2\}}.$$

Further, there exist values for ω_1, ω_2, α_1 and α_2 for which both inequalities are tight up to an arbitrary small constant. The values will adhere the same structure as the lower bound example given in Subsection 6.4.2. Finally, if both n_1 and n_2 tend to infinity then the above ratio approaches 2 for suitable choices of ω_1, ω_2, α_1 and α_2.

Appendix

6.A Postponed proofs of Subsection 6.4.2

This appended section will show the full technical proof of Theorem 6.4.7 for for which only an intuitive argument was provided in Subsection 6.4.2.

Theorem 6.4.7. *There exist parameter settings $\boldsymbol{n} > \mathbf{0}$, $\boldsymbol{\omega} > \mathbf{0}$ and $\boldsymbol{\alpha} > \mathbf{1}$, such that, for any $\epsilon > 0$*

$$\frac{\mathbf{E}[\Pi^{\ell}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})]}{\mathbf{E}[\Pi^{*}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})]} > 1 + \sqrt{m-1} - \epsilon.$$

I will present a sequence of results which slowly build up to show the final proof of Theorem 6.4.7. First, I present two properties showing the limit of some quantity when α_1 tends to infinity.

Property 6.A.1. *Let $\frac{\omega_1}{\alpha_1 - 1} = 1 - \sqrt{\frac{1}{\alpha_1}}$. Then,*

$$\lim_{\alpha_1 \to \infty} (\alpha_1 + n_1 - 1)^{n_1 - 1} \left(\frac{\omega_1}{\alpha_1}\right)^{\alpha_1} = 0 \quad \text{and} \quad \lim_{\alpha_1 \to \infty} \frac{\omega_1 - 1}{\alpha_1 + n_1 - 2} = \lim_{\alpha_1 \to \infty} \frac{\omega_1}{\alpha_1 - 1} = 1.$$

Proof. Set $\omega_1 = (\alpha_1 - 1)\left(1 - \sqrt{\frac{1}{\alpha_1}}\right) < \alpha_1 - \sqrt{\alpha_1}$. Then,

$$\left(\frac{\omega_1}{\alpha_1}\right)^{\alpha_1} < \left(1 - \sqrt{\frac{1}{\alpha_1}}\right)^{\alpha_1} = \left(\left(\frac{\sqrt{\alpha_1} - 1}{\sqrt{\alpha_1}}\right)^{\sqrt{\alpha_1}}\right)^{\sqrt{\alpha_1}} < \left(\left(\frac{\sqrt{\alpha_1} + 1}{\sqrt{\alpha_1}}\right)^{\sqrt{\alpha_1}}\right)^{-\sqrt{\alpha_1}}.$$

As $\lim_{\alpha_1 \to \infty} \left(\frac{\sqrt{\alpha_1}+1}{\sqrt{\alpha_1}}\right)^{\sqrt{\alpha_1}} = e$, it follows that

$$\lim_{\alpha_1 \to \infty} (\alpha_1 + n_1 - 1)^{n_1 - 1} \left(\frac{\omega_1}{\alpha_1}\right)^{\alpha_1} < \lim_{\alpha_1 \to \infty} (\alpha_1 + n_1 - 1)^{n_1 - 1} (e)^{-\sqrt{\alpha_1}} = 0,$$

yielding the first statement. Further, for the second statement,

$$\lim_{\alpha_1 \to \infty} \frac{\omega_1}{\alpha_1 - 1} = \lim_{\alpha_1 \to \infty} 1 - \sqrt{\frac{1}{\alpha_1}} = 1, \text{ such that}$$
$$\lim_{\alpha_1 \to \infty} \frac{\omega_1 - 1}{\alpha_1 + n_1 - 2} = \lim_{\alpha_1 \to \infty} \frac{(\alpha_1 - 1)\left(1 - \sqrt{\frac{1}{\alpha_1}}\right) - 1}{\alpha_1 + n_1 - 2} \lim_{\alpha_1 \to \infty} \frac{(\alpha_1 - 1) - 1}{\alpha_1 + n_1 - 2} = 1.$$

□

The next property provides a lower bound on the value of ℓ-SEPT given that a single job from job class J_1 has already been processed.

Property 6.A.2. *Let* $\frac{\omega_1}{\alpha_1-1} < \frac{\omega_2}{\alpha_2-1} = \ldots = \frac{\omega_m}{\alpha_m-1} = 1$ *and* $n_1 \geq 1$. *Then,*

$$\begin{aligned}
&\int_0^\infty \mathbf{E}[\Pi^\ell(\boldsymbol{n}, \boldsymbol{\omega} + x_1^1 \boldsymbol{e}_1, \boldsymbol{\alpha} + \boldsymbol{e}_1)] f_1^0(x_1^1)\, \partial x_1^1 \\
&> \frac{n_1(n_1+1)}{2} \frac{\omega_1 - 1}{\alpha_1 + n_1 - 1} + \frac{(n_2-1)n_2}{2} + \sum_{i=3}^m \frac{n_i(n_i+1)}{2} + \left(\left(n_2 + \sum_{i=2}^m n_1 n_i \right) \right. \\
&\left. * \int_0^{\alpha_1-\omega_1} \cdots \int_0^{\alpha_1+n_1-1-\omega_1-x_1^1-\ldots-x_1^{n_1-1}} \frac{\omega_1 - 1}{\alpha_1 + n_1 - 1} f_1^{n_1-1}(x_1^{n_1})\, \partial x_1^{n_1} \cdots f_1^0(x_1^1)\, \partial x_1^1 \right). \qquad (6.10)
\end{aligned}$$

Proof. As $\frac{\omega_1}{\alpha_1-1} < 1$ it is that $\omega_1 < \alpha_1 - 1 < \alpha_1$, yielding in combination with Property 6.4.1 that

$$\int_0^\infty \min\left\{ \frac{\omega_1 + x_1^1}{\alpha_1}, 1 \right\} f_1^0(x_1^1)\, \partial x_1^1 = \frac{\omega_1}{\alpha_1 - 1} - \frac{1}{\alpha_1 - 1} \left(\frac{\omega_1}{\alpha_1} \right)^{\alpha_1} > \frac{\omega_1 - 1}{\alpha_1 - 1}. \qquad (6.11)$$

Therefore,

$$\begin{aligned}
&\int_0^\infty \mathbf{E}[\Pi^\ell(\boldsymbol{n}, \boldsymbol{\omega} + x_1^1 \boldsymbol{e}_1, \boldsymbol{\alpha} + \boldsymbol{e}_1)] f_1^0(x_1^1)\, \partial x_1^1 \\
&= \int_0^{\alpha_1-\omega_1} \int_0^\infty \mathbf{E}[\Pi^\ell(\boldsymbol{n} - \boldsymbol{e}_1, \boldsymbol{\omega} + (x_1^1 + x_1^2)\boldsymbol{e}_1, \boldsymbol{\alpha} + 2\boldsymbol{e}_1)] f_1^1(x_1^2)\, \partial x_1^2 \cdot f_1^0(x_1^1)\, \partial x_1^1 \\
&+ \int_{\alpha_1-\omega_1}^\infty \int_0^\infty \mathbf{E}[\Pi^\ell(\boldsymbol{n} - \boldsymbol{e}_2, \boldsymbol{\omega} + x_1^1 \boldsymbol{e}_1 + x_2^1 \boldsymbol{e}_2, \boldsymbol{\alpha} + \boldsymbol{e}_1 + \boldsymbol{e}_2)] f_2^0(x_2^1)\, \partial x_2^1 \cdot f_1^0(x_1^1)\, \partial x_1^1 \\
&+ \int_0^{\alpha_1-\omega_1} \left(\frac{\omega_1 + x_1^1}{\alpha_1} \sum_{i=1}^m n_i \right) f_1^0(x_1^1)\, \partial x_1^1 + \int_{\alpha_1-\omega_1}^\infty \left(\frac{\omega_2}{\alpha_2 - 1} \sum_{i=1}^m n_i \right) f_1^0(x_1^1)\, \partial x_1^1 \\
&> \left(\sum_{i=1}^m n_i \right) \left(\int_0^{\alpha_1-\omega_1} \frac{\omega_1 + x_1^1}{\alpha_1} f_1^0(x_1^1)\, \partial x_1^1 + \int_{\alpha_1-\omega_1}^\infty 1 \cdot f_1^0(x_1^1)\, \partial x_1^1 \right) \\
&+ \int_0^{\alpha_1-\omega_1} \int_0^\infty \mathbf{E}[\Pi^\ell(\boldsymbol{n} - \boldsymbol{e}_1, \boldsymbol{\omega} + (x_1^1 + x_1^2)\boldsymbol{e}_1, \boldsymbol{\alpha} + 2\boldsymbol{e}_1)] f_1^1(x_1^2)\, \partial x_1^2 \cdot f_1^0(x_1^1)\, \partial x_1^1 \\
&+ \int_{\alpha_1-\omega_1}^\infty \left(\frac{n_1(n_1+1)}{2} \frac{\omega_1 + x_1^1}{\alpha_1} + \frac{(n_2-1)n_2}{2} \frac{\omega_2}{\alpha_2 - 1} + \sum_{i=3}^m \frac{n_i(n_i+1)}{2} \frac{\omega_i}{\alpha_i - 1} \right) f_1^0(x_1^1)\, \partial x_1^1 \\
&> \int_0^{\alpha_1-\omega_1} \int_0^\infty \mathbf{E}[\Pi^\ell(\boldsymbol{n} - \boldsymbol{e}_1, \boldsymbol{\omega} + (x_1^1 + x_1^2)\boldsymbol{e}_1, \boldsymbol{\alpha} + 2\boldsymbol{e}_1)] f_1^1(x_1^2)\, \partial x_1^2 \cdot f_1^0(x_1^1)\, \partial x_1^1 \\
&+ \int_{\alpha_1-\omega_1}^\infty \left(\frac{n_1(n_1+1)}{2} \frac{\omega_1 + x_1^1}{\alpha_1} + \frac{(n_2-1)n_2}{2} \frac{\omega_2}{\alpha_2 - 1} + \sum_{i=3}^m \frac{n_i(n_i+1)}{2} \frac{\omega_i}{\alpha_i - 1} \right) f_1^0(x_1^1)\, \partial x_1^1 \\
&+ \frac{\omega_1 - 1}{\alpha_1 - 1} \sum_{i=1}^m n_i, \qquad (6.12)
\end{aligned}$$

where the first inequality follows from Property 6.3.2 and the second inequality from (6.11). Using the derived inequality above, I show the property in (6.10) by induction over n_1. In case $n_1 = 1$, the first and second inequality below follow from inequality (6.12) and Property 6.3.2 respectively, such that

$$\int_0^\infty \mathbf{E}[\Pi^\ell(\boldsymbol{n}, \boldsymbol{\omega} + x_1^1 \boldsymbol{e}_1, \boldsymbol{\alpha} + \boldsymbol{e}_1)] f_1^0(x_1^1)\, \partial x_1^1$$

$$
\begin{aligned}
&> \left(1+\sum_{i=2}^{m} n_i\right)\frac{\omega_1-1}{\alpha_1-1} \\
&+ \int_{\alpha_1-\omega_1}^{\infty}\left(\frac{\omega_1+x_1}{\alpha_1}+\frac{(n_2-1)n_2}{2}\frac{\omega_2}{\alpha_2-1}+\sum_{i=3}^{m}\frac{n_i(n_i+1)}{2}\frac{\omega_i}{\alpha_i-1}\right) f_1^0(x_1^1)\,\partial x_1^1 \\
&+ \int_0^{\alpha_1-\omega_1}\int_0^{\infty} \mathbf{E}[\Pi^{\ell}(\boldsymbol{n}-n_1\boldsymbol{e}_1,\boldsymbol{\omega}+(x_1^1+x_1^2)\boldsymbol{e}_1,\boldsymbol{\alpha}+2\boldsymbol{e}_1)] f_1^1(x_1^2)\,\partial x_1^2\cdot f_1^0(x_1^1)\,\partial x_1^1 \\
&> \left(1+\sum_{i=2}^{m} n_i\right)\frac{\omega_1-1}{\alpha_1-1}+\frac{(n_2-1)n_2}{2}+\sum_{i=3}^{m}\frac{n_i(n_i+1)}{2} \\
&+ \int_{\alpha_1-\omega_1}^{\infty}\frac{\omega_1+x_1^1}{\alpha_1} f_1^0(x_1^1)\,\partial x_1^1+\int_0^{\alpha_1-\omega_1} n_2 f_1^0(x_1^1)\,\partial x_1^1 \\
&> \frac{\omega_1-1}{\alpha_1}+\frac{(n_2-1)n_2}{2}+\sum_{i=3}^{m}\frac{n_i(n_i+1)}{2}+\left(n_2+\sum_{i=2}^{m} n_i\right)\int_0^{\alpha_1-\omega_1}\frac{\omega_1-1}{\alpha_1} f_1^0(x_1^1)\,\partial x_1^1,
\end{aligned}
$$

where the last inequality is due to $1>\frac{\omega_1-1}{\alpha_1-1}>\frac{\omega_1-1}{\alpha_1}$. Next, assume that (6.10) is correct for n_1. I will verify below that the (6.10) also holds true for n_1+1.

$$
\begin{aligned}
&\int_0^{\infty}\mathbf{E}[\Pi^{\ell}(\boldsymbol{n}+\boldsymbol{e}_1,\boldsymbol{\omega}+x_1^1\boldsymbol{e}_1,\boldsymbol{\alpha}+\boldsymbol{e}_1)] f_1^0(x_1^1)dx_1^1 \\
&\overset{(6.12)}{>} \left(n_1+1+\sum_{i=2}^{m} n_i\right)\frac{\omega_1-1}{\alpha_1-1} \\
&+ \int_{\alpha_1-\omega_1}^{\infty}\left(\frac{(n_1+1)(n_1+2)}{2}\frac{\omega_1+x_1^1}{\alpha_1}+\frac{(n_2-1)n_2}{2}\frac{\omega_2}{\alpha_2-1}+\sum_{i=3}^{m}\frac{n_i(n_i+1)}{2}\frac{\omega_i}{\alpha_i-1}\right) f_1^0(x_1^1)\,\partial x_1^1 \\
&+ \int_0^{\alpha_1-\omega_1}\int_0^{\infty} \mathbf{E}[\Pi^{\ell}(\boldsymbol{n},\boldsymbol{\omega}+(x_1^1+x_1^2)\boldsymbol{e}_1,\boldsymbol{\alpha}+2\boldsymbol{e}_1)] f_1^1(x_1^2)\,\partial x_1^2\cdot f_1^0(x_1^1)\,\partial x_1^1 \\
&> \left(n_1+1+\sum_{i=2}^{m} n_i\right)\frac{\omega_1-1}{\alpha_1-1} \\
&+ \int_{\alpha_1-\omega_1}^{\infty}\left(\frac{(n_1+1)(n_1+2)}{2}\frac{\omega_1+x_1^1}{\alpha_1}+\frac{(n_2-1)n_2}{2}\frac{\omega_2}{\alpha_2-1}+\sum_{i=3}^{m}\frac{n_i(n_i+1)}{2}\frac{\omega_i}{\alpha_i-1}\right) f_1^0(x_1^1)\,\partial x_1^1 \\
&+ \int_0^{\alpha_1-\omega_1}\left(\frac{n_1(n_1+1)}{2}\frac{\omega_1+x_1^1-1}{\alpha_1+n_1}+\frac{(n_2-1)n_2}{2}+\sum_{i=3}^{m}\frac{n_i(n_i+1)}{2}\right) f_1^0(x_1^1)\,\partial x_1^1 \\
&+ \left(n_2+\sum_{i=2}^{m} n_1 n_i\right)\int_0^{\alpha_1-\omega_1}\cdots\int_0^{\alpha_1+n_1-\omega_1-x_1^1-\ldots-x_1^{n_1}}\frac{\omega_1+x_1^1-1}{\alpha_1+n_1} f_1^{n_1}(x_1^{n_1+1})\,\partial x_1^{n_1+1}\cdots f_1^0(x_1^1)\,\partial x_1^1 \\
&> \left(n_1+1+\sum_{i=2}^{m} n_i\right)\frac{\omega_1-1}{\alpha_1+n_1}+\frac{n_2(n_2-1)}{2}+\sum_{i=3}^{m}\frac{n_i(n_i+1)}{2} \\
&+ \int_{\alpha_1-\omega_1}^{\infty}\left(\frac{(n_1+1)(n_1+2)}{2}\frac{\omega_1-1}{\alpha_1+n_1}\right) f_1^0(x_1^1)\,\partial x_1^1 \\
&+ \int_0^{\alpha_1-\omega_1}\left(\frac{n_1(n_1+1)}{2}\frac{\omega_1-1}{\alpha_1+n_1}\right) f_1^0(x_1^1)\,\partial x_1^1
\end{aligned}
$$

$$
\begin{aligned}
&+\left(n_2+\sum_{i=2}^{m} n_1 n_i\right)\int_0^{\alpha_1-\omega_1}\cdots\int_0^{\alpha_1+n_1-\omega_1-x_1^1-\ldots-x_1^{n_1}} \frac{\omega_1-1}{\alpha_1+n_1}\, f_1^{n_1}(x_1^{n_1+1})\,\partial x_1^{n_1+1}\cdots f_1^0(x_1^1)\,\partial x_1^1\\
&>\frac{n_2(n_2-1)}{2}+\sum_{i=3}^{m}\frac{n_i(n_i+1)}{2}+\frac{(n_1+1)(n_1+2)}{2}\frac{\omega_1-1}{\alpha_1+n_1}\\
&+\left(n_2+\sum_{i=2}^{m}(n_1+1)n_i\right)\int_0^{\alpha_1-\omega_1}\cdots\int_0^{\alpha_1+n_1-\omega_1-x_1^1-\ldots-x_1^{n_1}} \frac{\omega_1-1}{\alpha_1+n_1}\, f_1^{n_1}(x_1^{n_1+1})\,\partial x_1^{n_1+1}\cdots f_1^0(x_1^1)\,\partial x_1^1,
\end{aligned}
$$

where the second inequality follows from the induction step. This is allowed, since inside the second integral on the right hand side $x_1 < \alpha_1 - \omega_1$ such that $\frac{\omega_1+x_1^1}{\alpha_1} < 1 = \frac{\omega_2}{\alpha_2-1} = \ldots = \frac{\omega_m}{\alpha_m-1}$. □

The next property lower bounds the last term in the right hand side of (6.10) with a more convenient closed formula.

Property 6.A.3. *Let* $\frac{\omega_1}{\alpha_1-1} < \frac{\omega_2}{\alpha_2-1} = \ldots = \frac{\omega_m}{\alpha_m-1} = 1$. *Then, for any* $n_1 \geq 1$,

$$
\begin{aligned}
&\int_0^{\alpha_1-\omega_1}\cdots\int_0^{\alpha_1+n_1-1-\omega_1-x_1^1-\ldots-x_1^{n_1-1}} 1\cdot f_1^{n_1-1}(x_1^{n_1})\,\partial x_1^{n_1}\cdots f_1^0(x_1^1)\,\partial x_1^1\\
&\qquad\geq 1-(\alpha_1+n_1-1)^{n_1-1}\left(\frac{\omega_1}{\alpha_1}\right)^{\alpha_1}.
\end{aligned}
$$

Proof. I provide a proof by induction on n_1. In case $n_1 = 1$, it is that

$$
\int_0^{\alpha_1-\omega_1} 1\cdot\, f_1^0(x_1^1)\,\partial x_1^1 = \int_0^{\alpha_1-\omega_1}\frac{\alpha_1\cdot\omega_1^{\alpha_1}}{(\omega_1+x_1^1)^{\alpha_1+1}}\,\partial x_1^1 = 1-\left(\frac{\omega_1}{\alpha_1}\right)^{\alpha_1}. \tag{6.13}
$$

Now, assume that the statement is correct for all $n \leq n_1$. I verify it also holds for n_1+1:

$$
\begin{aligned}
&\int_0^{\alpha_1-\omega_1}\cdots\int_0^{\alpha_1+n_1-\omega_1-x_1^1-\ldots-x_1^{n_1}} 1\cdot f_1^{n_1}(x_1^{n_1+1})\,\partial x_1^{n_1+1}\cdots f_1^1(x_1^2)\,\partial x_1^2\cdot f_1^0(x_1^1)\,\partial x_1^1\\
&\overset{\text{(induction)}}{\geq}\int_0^{\alpha_1-\omega_1}\left(1-(\alpha_1+n_1)^{n_1-1}\left(\frac{\omega_1+x_1^1}{\alpha_1+1}\right)^{\alpha_1+1}\right) f_1^0(x_1^1)\,\partial x_1^1\\
&\overset{(6.13)}{=}1-\left(\frac{\omega_1}{\alpha_1}\right)^{\alpha_1}-\int_0^{\alpha_1-\omega_1}(\alpha_1+n_1)^{n_1-1}\left(\frac{\omega_1+x_1^1}{\alpha_1+1}\right)^{\alpha_1+1}\frac{\alpha_1\omega_1^{\alpha_1}}{(\omega_1+x_1^1)^{\alpha_1+1}}\,\partial x_1\\
&=1-\left(\frac{\omega_1}{\alpha_1}\right)^{\alpha_1}-(\alpha_1+n_1)^{n_1-1}\frac{\alpha_1}{\alpha_1+1}\left(\frac{\omega_1}{\alpha_1+1}\right)^{\alpha_1}(\alpha_1-\omega_1)\\
&>1-\left[1+\alpha_1(\alpha_1+n_1)^{n_1-1}\right]\left(\frac{\omega_1}{\alpha_1}\right)^{\alpha_1} > 1-(\alpha_1+n_1)^{n_1}\left(\frac{\omega_1}{\alpha_1}\right)^{\alpha_1}.
\end{aligned}
$$

□

The following property provides a lower bound on the expected value of ℓ-SEPT and follows from the three preceding properties.

Lemma 6.A.4. *For any $\boldsymbol{n} \geq \mathbf{0}$, there exist parameter settings $\boldsymbol{\omega} > \mathbf{0}$, and $\boldsymbol{\alpha} > \mathbf{1}$ such that $\omega_1/(\alpha_1 - 1) < \omega_2/(\alpha_2 - 1) = \ldots = \omega_m/(\alpha_m - 1) = 1$, and*

$$\mathbf{E}[\Pi^{\ell}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] > \sum_{i=1}^{m} \frac{n_i(n_i+1)}{2} + \sum_{i=2}^{m} n_1 n_i - \epsilon,$$

for any $\epsilon > 0$.

Proof. For arbitrary $\boldsymbol{\alpha} > \mathbf{1}$, set $\omega_1 = (\alpha_1 - 1)\left(1 - \sqrt{\frac{1}{\alpha_1}}\right)$ and $\omega_i = \omega_{i'} = \alpha_i - 1$ for all $i, i' \geq 2$. Then,

$$\begin{aligned}
\mathbf{E}[\Pi^{\ell}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] &= \frac{\omega_1}{\alpha_1 - 1} \sum_{i=1}^{m} n_i + \int_0^{\infty} \mathbf{E}[\Pi^{\ell}(\boldsymbol{n} - \boldsymbol{e}_1, \boldsymbol{\omega} + x_1^1 \boldsymbol{e}_1, \boldsymbol{\alpha} + \boldsymbol{e}_1)] f_1^0(x_1^1)\, \partial x_1^1 \\
&> \frac{\omega_1}{\alpha_1 - 1} \sum_{i=1}^{m} n_i + \frac{(n_1 - 1)n_1}{2} \frac{\omega_1 - 1}{\alpha_1 + n_1 - 2} + \frac{(n_2 - 1)n_2}{2} + \sum_{i=3}^{m} \frac{n_i(n_i+1)}{2} \\
&\quad + \left(n_2 + \sum_{i=2}^{m} (n_1 - 1)n_i\right) \left(1 - (\alpha_1 + n_1 - 2)^{n_1 - 2} \left(\frac{\omega_1}{\alpha_1}\right)^{\alpha_1}\right) \frac{\omega_1 - 1}{\alpha_1 + n_1 - 2}, \qquad (6.14)
\end{aligned}$$

where the inequality is due to Properties 6.A.2 and 6.A.3. Letting α_1 tend to infinity, it follows from (6.14) and Property 6.A.1, that

$$\begin{aligned}
&\lim_{\alpha_1 \to \infty} \mathbf{E}[\Pi^{\ell}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] \\
&\quad > \sum_{i=1}^{m} n_i + \frac{(n_1 - 1)n_1}{2} + \frac{(n_2 - 1)n_2}{2} + \sum_{i=3}^{m} \frac{n_i(n_i+1)}{2} + \left(n_2 + \sum_{i=2}^{m} (n_1 - 1)n_i\right) \\
&\quad = \sum_{i=1}^{m} \frac{n_i(n_i+1)}{2} + \sum_{i=2}^{m} n_1 n_i.
\end{aligned}$$

Hence, for any $\epsilon > 0$, there exists an $\alpha^* > 1$ such that for all $\alpha_1 > \alpha^*$

$$\mathbf{E}[\Pi^{\ell}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] > \sum_{i=1}^{m} \frac{n_i(n_i+1)}{2} + \sum_{i=2}^{m} n_1 n_i - \epsilon,$$

where $\omega_1 = (\alpha_1 - 1)\left(1 - \sqrt{\frac{1}{\alpha_1}}\right)$ and $\omega_2 = \ldots = \omega_m = \alpha_2 - 1 = \ldots = \alpha_m - 1$. □

The preceding result, in combination with Lemma 6.4.5, shows the lower bound on performance guarantee of ℓ-SEPT.

Proof of Theorem 6.4.7. By Lemmata 6.4.5 and 6.A.4, I have that there exist values for $\boldsymbol{\omega} > \mathbf{0}$ and $\boldsymbol{\alpha} > \mathbf{1}$ such that

$$\frac{\mathbf{E}[\Pi^{\ell}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})]}{\mathbf{E}[\Pi^{*}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})]} > \frac{\sum_{i=1}^{m} n_i(n_i+1) + 2\sum_{i=2}^{m} n_1 n_i}{\sum_{i=1}^{m} n_i(n_i+1) + 2m\sum_{i=1}^{m} n_i} - \epsilon$$

for any $\epsilon > 0$. Then, by setting $n_2 = \ldots = n_m = n$, $n_1 = n\sqrt{m-1}$ and letting n tend to infinity, for any $\epsilon > 0$ it holds true that

$$\lim_{n \to \infty} \frac{\mathbf{E}[\Pi^{\ell}(n_1, n_2, \boldsymbol{\omega}, \boldsymbol{\alpha})]}{\mathbf{E}[\Pi^{*}(n_1, n_2, \boldsymbol{\omega}, \boldsymbol{\alpha})]} > \lim_{n \to \infty} \frac{(m-1)(2n^2+n) + n\sqrt{m-1} + 2(m-1)n^2\sqrt{m-1}}{(m-1)(2n^2+n) + n\sqrt{m-1} + 2mn\sqrt{m-1} + 2m(m-1)n} - \epsilon$$

$$= \frac{2(m-1) + 2(m-1)\sqrt{m-1)}}{2(m-1)} - \epsilon = 1 + \sqrt{m-1} - \epsilon.$$

□

6.B Postponed proofs of Section 6.5

This second appended section provides the postponed proofs of Section 6.5, that is, the proofs of Lemmas 6.5.1, 6.5.2 and 6.5.3.

Lemma 6.5.1. *For any $\boldsymbol{n} \geq \mathbf{0}$, $\boldsymbol{\omega} > \mathbf{0}$ and $\boldsymbol{\alpha} > \mathbf{1}$ with $\frac{\omega_1}{\alpha_1 - 1} \leq \frac{\omega_2}{\alpha_2 - 1} \leq \ldots \leq \frac{\omega_m}{\alpha_m - 1}$,*

$$\mathbf{E}[\Pi^*(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] \geq \sum_{i=1}^{m} \frac{n_i(n_i+1)}{2} \frac{\omega_i}{\alpha_i - 1} + \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} n_i n_j \, \xi(i,j) \, \frac{\omega_i}{\alpha_i - 1}, \text{ where}$$

$$\xi(i,j) = 1 - \frac{\left(\frac{\omega_i}{\omega_i + \omega_j}\right)^{\alpha_i - 1} \left(\frac{\omega_j}{\omega_i + \omega_j}\right)^{\alpha_j - 1}}{(\alpha_i + \alpha_j - 1) \cdot B(\alpha_i, \alpha_j)} \in (0,1)\,,$$

and $B(\alpha_i, \alpha_j) = \int_0^1 t^{\alpha_i - 1}(1-t)^{\alpha_j - 1} \, \partial t$ denotes a Beta function.

Proof. Let $Y = \min\left\{\frac{1}{\Theta_i}, \frac{1}{\Theta_j}\right\}$ for some $i \neq j$. Then,

$$1 - F_Y(y) = 1 - \mathbf{Pr}[Y \leq y] = \mathbf{Pr}\left[\frac{1}{\Theta_i} > y\right] \mathbf{Pr}\left[\frac{1}{\Theta_j} > y\right]$$
$$= \left(1 - F_{\Theta_i^{-1}}(y)\right)\left(1 - F_{\Theta_j^{-1}}(y)\right) = \left(1 - F_{\Theta_i^{-1}}(y)\right) + \left(1 - F_{\Theta_j^{-1}}(y)\right) - \left(1 - F_{\Theta_i^{-1}}(y) F_{\Theta_j^{-1}}(y)\right).$$

such that

$$\mathbf{E}\left[\min\left\{\frac{1}{\Theta_i}, \frac{1}{\Theta_j}\right\}\right] = \int_0^\infty (1 - F_Y(y)) \, \partial y \tag{6.15}$$
$$= \frac{\omega_i}{\alpha_i - 1} + \frac{\omega_j}{\alpha_j - 1} + \int_0^\infty \left(1 - F_{\Theta_i^{-1}}(y) F_{\Theta_j^{-1}}(y)\right) \partial y,$$

where

$$\int_0^\infty \left(1 - F_{\Theta_i^{-1}}(y) F_{\Theta_j^{-1}}(y)\right) \partial y$$
$$= -\frac{\omega_i}{\alpha_i - 1} \left(\frac{\omega_i}{\omega_j}\right)^{\alpha_i - 1} \frac{\Gamma(\alpha_i + \alpha_j - 1)}{\Gamma(\alpha_i)\Gamma(\alpha_j)} {}_2F_1\left(\alpha_i + \alpha_j - 1, \alpha_i - 1, \alpha_i, -\left(\frac{\omega_i}{\omega_j}\right)\right)$$
$$- \frac{\omega_j}{\alpha_j - 1} \left(\frac{\omega_j}{\omega_i}\right)^{\alpha_j - 1} \frac{\Gamma(\alpha_i + \alpha_j - 1)}{\Gamma(\alpha_i)\Gamma(\alpha_j)} {}_2F_1\left(\alpha_i + \alpha_j - 1, \alpha_j - 1, \alpha_j, -\left(\frac{\omega_j}{\omega_i}\right)\right)$$
$$= -\frac{\omega_i}{\alpha_i - 1} I_{\omega_i/(\omega_i + \omega_j)}(\alpha_i - 1, \alpha_j) - \frac{\omega_j}{\alpha_j - 1} I_{\omega_j/(\omega_i + \omega_j)}(\alpha_j - 1, \alpha_i). \tag{6.16}$$

In the above, the first equality follows from solving the integrals, and the last one from the definition of the regularized incomplete beta function

$$0 \leq I_z(a,b) = \frac{B(z;a,b)}{B(a,b)} = \frac{\int_0^z t^{a-1}(1-t)^{b-1} \, \partial t}{\int_0^\infty t^{a-1}(1-t)^{b-1} \, \partial t} \leq 1 \tag{6.17}$$

for all $0 \leq z \leq 1$ and $a, b > 0$. Also ${}_2F_1(a, b, c, z)$ denotes a hypergeometric function. When $|z| < 1$, the function can be expressed as follows

$$ {}_2F_1(a, b, c, z) = 1 + \sum_{n=1}^{\infty} \frac{(a)_n (b)_n}{(c)_n} \frac{z^n}{n!}, $$

where $(a)_n = a * (a+1) * \cdots * (a+n-1)$. Note that

$$ I_z(a, b) = I_z(a+1, b) + \frac{z^a (1-z)^b}{aB(a, b)}, \tag{6.18} $$

$$ I_z(a, b) + I_{1-z}(b, a) = 1, \tag{6.19} $$

and

$$ B(a, b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}. \tag{6.20} $$

By (6.8) and the implied order on the expected processing times of the classes, i. e., $\frac{\omega_1}{\alpha_1 - 1} \leq \frac{\omega_2}{\alpha_2 - 1} \leq \ldots \leq \frac{\omega_m}{\alpha_m - 1}$, it is that

$$
\begin{aligned}
\mathbf{E}[\Pi^*(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] &\geq \sum_{i=1}^{m} \frac{n_i(n_i+1)}{2} \frac{\omega_i}{\alpha_i - 1} + \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} n_i n_j \, \mathbf{E}\left[\min\left\{\frac{1}{\Theta_i}, \frac{1}{\Theta_j}\right\}\right] \\
&\overset{(6.15)}{=} \sum_{i=1}^{m} \frac{n_i(n_i+1)}{2} \frac{\omega_i}{\alpha_i - 1} + \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} (n_i n_j \\
&\quad * \left(\frac{\omega_i}{\alpha_i - 1} + \frac{\omega_j}{\alpha_j - 1} - \frac{\omega_i}{\alpha_i - 1} I_{\omega_i/(\omega_i+\omega_j)}(\alpha_i - 1, \alpha_j) - \frac{\omega_j}{\alpha_j - 1} I_{\omega_j/(\omega_i+\omega_j)}(\alpha_j - 1, \alpha_i) \right) \Big) \\
&\overset{(6.17)}{>} \sum_{i=1}^{m} \frac{n_i(n_i+1)}{2} \frac{\omega_i}{\alpha_i - 1} \\
&\quad + \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} n_i n_j \left(2 - I_{\omega_i/(\omega_i+\omega_j)}(\alpha_i - 1, \alpha_j) - I_{\omega_j/(\omega_i+\omega_j)}(\alpha_j - 1, \alpha_i)\right) \frac{\omega_i}{\alpha_i - 1} \\
&\overset{(6.18)}{=} \sum_{i=1}^{m} \frac{n_i(n_i+1)}{2} \frac{\omega_i}{\alpha_i - 1} + \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} n_i n_j \left(2 - \left(I_{\omega_i/(\omega_i+\omega_j)}(\alpha_i, \alpha_j) + \frac{\left(\frac{\omega_i}{\omega_i+\omega_j}\right)^{\alpha_i - 1} \left(\frac{\omega_j}{\omega_i+\omega_j}\right)^{\alpha_j}}{B(\alpha_i - 1, \alpha_j)(\alpha_i - 1)} \right) \right. \\
&\quad \left. - \left(I_{\omega_j/(\omega_i+\omega_j)}(\alpha_j, \alpha_i) + \frac{\left(\frac{\omega_j}{\omega_i+\omega_j}\right)^{\alpha_j - 1} \left(\frac{\omega_i}{\omega_i+\omega_j}\right)^{\alpha_i}}{B(\alpha_j - 1, \alpha_i)(\alpha_j - 1)} \right) \right) \frac{\omega_i}{\alpha_i - 1}.
\end{aligned}
$$

The proof is concluded by rewriting the last line using (6.19), (6.20), and $\Gamma(a) = (a-1)\Gamma(a-1)$. □

Lemma 6.5.2. *For any $\boldsymbol{n} \geq \mathbf{0}, \boldsymbol{\omega} > \mathbf{0}$ and $\boldsymbol{\alpha} > \mathbf{1}$,*

$$ \mathbf{E}[\Pi^\ell(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] = \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \mathbf{E}[\Pi^\ell(n_i \boldsymbol{e}_i + n_j \boldsymbol{e}_j, \boldsymbol{\omega}, \boldsymbol{\alpha})] - (m-2) \sum_{i=1}^{m} \mathbf{E}[\Pi^\ell(n_i \boldsymbol{e}_i, \boldsymbol{\omega}, \boldsymbol{\alpha})]. $$

Proof. The proof is by induction. The statement trivially holds if $\boldsymbol{n} = n_h \boldsymbol{e}_h$ for any $h \in [m]$, then

$$\sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \mathbf{E}[\Pi^{\ell}(n_i \boldsymbol{e}_i + n_j \boldsymbol{e}_j, \boldsymbol{\omega}, \boldsymbol{\alpha})] - (m-2) \sum_{i=1}^{m} \mathbf{E}[\Pi^{\ell}(n_i \boldsymbol{e}_i, \boldsymbol{\omega}, \boldsymbol{\alpha})]$$
$$= (m-1)\, \mathbf{E}[\Pi^{\ell}(n_h \boldsymbol{e}_h, \boldsymbol{\omega}, \boldsymbol{\alpha})] - (m-2)\, \mathbf{E}[\Pi^{\ell}(n_h \boldsymbol{e}_h, \boldsymbol{\omega}, \boldsymbol{\alpha})] = \mathbf{E}[\Pi^{\ell}(n_h \boldsymbol{e}_h, \boldsymbol{\omega}, \boldsymbol{\alpha})].$$

Consider two vectors $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^m$. We define $\boldsymbol{v} \prec \boldsymbol{w}$ when $v_i \leq w_i$ for all $i \in [m]$ and there exists some $i \in [m]$ such that $v_i < w_i$. Now assume that the statement in Equation (6.21) is true for all $\boldsymbol{n}' \prec \boldsymbol{n}$:

$$\mathbf{E}[\Pi^{\ell}(\boldsymbol{n}', \boldsymbol{\omega}, \boldsymbol{\alpha})] = \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \mathbf{E}[\Pi^{\ell}(n'_i \boldsymbol{e}_i + n'_j \boldsymbol{e}_j, \boldsymbol{\omega}, \boldsymbol{\alpha})] - (m-2) \sum_{i=1}^{m} \mathbf{E}[\Pi^{\ell}(n'_i \boldsymbol{e}_i, \boldsymbol{\omega}, \boldsymbol{\alpha})]. \tag{6.21}$$

The next step is to show that (6.21) also holds for $\boldsymbol{n}' = \boldsymbol{n}$. Without loss of generality, let $\frac{\omega_1}{\alpha_1 - 1} = \min_{i \in [m]} \frac{\omega_i}{\alpha_i - 1}$. Then,

$$\begin{aligned}
\mathbf{E}[\Pi^{\ell}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] &= \frac{\omega_1}{\alpha_1 - 1} \sum_{i=1}^{m} n_i + \int_0^{\infty} \mathbf{E}[\Pi^{\ell}(\boldsymbol{n} - \boldsymbol{e}_1, \boldsymbol{\omega} + x_1^1 \boldsymbol{e}_1, \boldsymbol{\alpha} + \boldsymbol{e}_1)] f_1^0(x_1^1)\, \partial x_1^1 \\
&\overset{(6.21)}{=} \frac{\omega_1}{\alpha_1 - 1} \sum_{i=1}^{m} n_i \\
&\quad + \int_0^{\infty} \left(\sum_{j=2}^{m} \mathbf{E}[\Pi^{\ell}((n_1 - 1)\boldsymbol{e}_1 + n_j \boldsymbol{e}_j, \boldsymbol{\omega} + x_1^1 \boldsymbol{e}_1, \boldsymbol{\alpha} + \boldsymbol{e}_1)] \right) f_1^0(x_1^1)\, \partial x_1^1 \\
&\quad + \int_0^{\infty} \left(\sum_{i=2}^{m-1} \sum_{j=i+1}^{m} \mathbf{E}[\Pi^{\ell}(n_i \boldsymbol{e}_i + n_j \boldsymbol{e}_j, \boldsymbol{\omega} + x_1^1 \boldsymbol{e}_1, \boldsymbol{\alpha} + \boldsymbol{e}_1)] \right) f_1^0(x_1^1)\, \partial x_1^1 \\
&\quad - (m-2) \int_0^{\infty} \mathbf{E}[\Pi^{\ell}((n_1 - 1)\boldsymbol{e}_1, \boldsymbol{\omega} + x_1^1 \boldsymbol{e}_1, \boldsymbol{\alpha} + \boldsymbol{e}_1)] f_1^0(x_1^1)\, \partial x_1^1 \\
&\quad - (m-2) \int_0^{\infty} \left(\sum_{i=2}^{m} \mathbf{E}[\Pi^{\ell}(n_i \boldsymbol{e}_i, \boldsymbol{\omega} + x_1^1 \boldsymbol{e}_1, \boldsymbol{\alpha} + \boldsymbol{e}_1)] \right) f_1^0(x_1^1)\, \partial x_1^1 \\
&= \left((m-2)n_1 + \sum_{i=1}^{m} n_i \right) \frac{\omega_1}{\alpha_1 - 1} - (m-2) \sum_{i=2}^{m} \mathbf{E}[\Pi^{\ell}(n_i \boldsymbol{e}_i, \boldsymbol{\omega}, \boldsymbol{\alpha})] \\
&\quad + \int_0^{\infty} \left(\sum_{j=2}^{m} \mathbf{E}[\Pi^{\ell}((n_1 - 1)\boldsymbol{e}_1 + n_j \boldsymbol{e}_j, \boldsymbol{\omega} + x_1^1 \boldsymbol{e}_1, \boldsymbol{\alpha} + \boldsymbol{e}_1)] \right) f_1^0(x_1^1)\, \partial x_1^1 \\
&\quad + \sum_{i=2}^{m-1} \sum_{j=i+1}^{m} \mathbf{E}[\Pi^{\ell}(n_i \boldsymbol{e}_i + n_j \boldsymbol{e}_j, \boldsymbol{\omega}, \boldsymbol{\alpha})] - (m-2)\, \mathbf{E}[\Pi^{\ell}(n_1 \boldsymbol{e}_1, \boldsymbol{\omega}, \boldsymbol{\alpha})] \\
&= \left((m-1)n_1 + \sum_{i=2}^{m} n_i \right) \frac{\omega_1}{\alpha_1 - 1} + \sum_{j=2}^{m} \left(\mathbf{E}[\Pi^{\ell}(n_1 \boldsymbol{e}_1 + n_j \boldsymbol{e}_j, \boldsymbol{\omega}, \boldsymbol{\alpha})] - (n_1 + n_j) \frac{\omega_1}{\alpha_1 - 1} \right) \\
&\quad + \sum_{i=2}^{m-1} \sum_{j=i+1}^{m} \mathbf{E}[\Pi^{\ell}(n_i \boldsymbol{e}_i + n_j \boldsymbol{e}_j, \boldsymbol{\omega}, \boldsymbol{\alpha})] - (m-2) \sum_{i=1}^{m} \mathbf{E}[\Pi^{\ell}(n_i \boldsymbol{e}_i, \boldsymbol{\omega}, \boldsymbol{\alpha})]
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{j=2}^{m} \mathbf{E}[\Pi^{\ell}(n_1\boldsymbol{e}_1 + n_j\boldsymbol{e}_j, \boldsymbol{\omega}, \boldsymbol{\alpha})] + \sum_{i=2}^{m-1}\sum_{j=i+1}^{m} \mathbf{E}[\Pi^{\ell}(n_i\boldsymbol{e}_i + n_j\boldsymbol{e}_j, \boldsymbol{\omega}, \boldsymbol{\alpha})] \\
&\quad - (m-2)\sum_{i=1}^{m} \mathbf{E}[\Pi^{\ell}(n_i\boldsymbol{e}_i, \boldsymbol{\omega}, \boldsymbol{\alpha})] \\
&= \sum_{i=1}^{m-1}\sum_{j=i+1}^{m} \mathbf{E}[\Pi^{\ell}(n_i\boldsymbol{e}_i + n_j\boldsymbol{e}_j, \boldsymbol{\omega}, \boldsymbol{\alpha})] - (m-2)\sum_{i=1}^{m} \mathbf{E}[\Pi^{\ell}(n_i\boldsymbol{e}_i, \boldsymbol{\omega}, \boldsymbol{\alpha})].
\end{aligned}
$$

□

Lemma 6.5.3. *For any $\boldsymbol{n} \geq \mathbf{0}, \boldsymbol{\omega} > \mathbf{0}$ and $\boldsymbol{\alpha} > \mathbf{1}$ with $\frac{\omega_1}{\alpha_1 - 1} \leq \frac{\omega_2}{\alpha_2 - 1} \leq \ldots \leq \frac{\omega_m}{\alpha_m - 1}$,*

$$
\begin{aligned}
\mathbf{E}[\Pi^{\ell}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] &\leq \sum_{i=1}^{m} \frac{n_i(n_i+1)}{2}\frac{\omega_i}{\alpha_i - 1} \\
&\quad + \sum_{i=1}^{m-1}\sum_{j=i+1}^{m} \left((n_i - 1)n_j \frac{\omega_i}{\alpha_i - 1}\,\zeta(i,j) \,+ n_j \frac{\omega_i}{\alpha_i - 1} \right), \\
\textit{where } \zeta(i,j) &= \left(1 - \frac{1}{\alpha_i}\left(\frac{\omega_i(\alpha_j - 1)}{\alpha_i \omega_j} \right)^{\alpha_i - 1} \right) \in (0,1)\,.
\end{aligned}
$$

Proof. Let $\boldsymbol{n} \geq \mathbf{0}, \boldsymbol{\omega} > \mathbf{0}$ and $\boldsymbol{\alpha} > \mathbf{1}$ be such that $\frac{\omega_1}{\alpha_1 - 1} \leq \frac{\omega_2}{\alpha_2 - 1} \leq \ldots \leq \frac{\omega_m}{\alpha_m - 1}$. By Lemma 6.5.2, it follows that

$$
\begin{aligned}
\mathbf{E}[\Pi^{\ell}(\boldsymbol{n}, \boldsymbol{\omega}, \boldsymbol{\alpha})] &= \sum_{i=1}^{m-1}\sum_{j=i+1}^{m} \mathbf{E}[\Pi^{\ell}(n_i\boldsymbol{e}_i + n_j\boldsymbol{e}_j, \boldsymbol{\omega}, \boldsymbol{\alpha})] - (m-2)\sum_{i=1}^{m} \mathbf{E}[\Pi^{\ell}(n_i\boldsymbol{e}_i, \boldsymbol{\omega}, \boldsymbol{\alpha})] \\
&= \sum_{i=1}^{m-1}\sum_{j=i+1}^{m} \int_0^{\infty} \mathbf{E}[\Pi^{\ell}((n_i - 1)\boldsymbol{e}_i + n_j\boldsymbol{e}_j, \boldsymbol{\omega} + x_i^1\boldsymbol{e}_i, \boldsymbol{\alpha} + \boldsymbol{e}_i)] f_i^0(x_i^1)\, \partial x_i^1 \\
&\quad \sum_{i=1}^{m-1}\sum_{j=i+1}^{m} (n_i + n_j)\frac{\omega_i}{\alpha_i - 1} - (m-2)\sum_{i=1}^{m} \frac{n_i(n_i+1)}{2}\frac{\omega_i}{\alpha_i - 1} \\
&\overset{(6.6)}{\leq} \sum_{i=1}^{m-1}\sum_{j=i+1}^{m} n_j \frac{\omega_i}{\alpha_i - 1} + \frac{n_i(n_i+1)}{2}\frac{\omega_i}{\alpha_i - 1} + \frac{n_j(n_j+1)}{2}\frac{\omega_j}{\alpha_j - 1} \\
&\quad + \sum_{i=1}^{m-1}\sum_{j=i+1}^{m} (n_i - 1)n_j \int_0^{\infty} \min\left\{ \frac{\omega_i + x_i^1}{\alpha_i}, \frac{\omega_j}{\alpha_j - 1} \right\} f_i^0(x_i^1)\, \partial x_i^1 \\
&\quad - (m-2)\sum_{i=1}^{m} \frac{n_i(n_i+1)}{2}\frac{\omega_i}{\alpha_i - 1} \\
&= \sum_{i=1}^{m} \frac{n_i(n_i+1)}{2}\frac{\omega_i}{\alpha_i - 1} + \sum_{i=1}^{m-1}\sum_{j=i+1}^{m} n_j \frac{\omega_i}{\alpha_i - 1} \\
&\quad + \sum_{i=1}^{m-1}\sum_{j=i+1}^{m} (n_i - 1)n_j \int_0^{\infty} \min\left\{ \frac{\omega_i + x_i^1}{\alpha_i}, \frac{\omega_j}{\alpha_j - 1} \right\} f_i^0(x_i^1)\, \partial x_i^1
\end{aligned}
$$

$$\overset{\text{Prop. 6.5.4}}{=} \sum_{i=1}^{m} \frac{n_i(n_i+1)}{2} \frac{\omega_i}{\alpha_i - 1} + \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \left((n_i - 1) n_j \frac{\omega_i}{\alpha_i - 1} \left(1 - \frac{1}{\alpha_i} \left(\frac{\omega_i(\alpha_j - 1)}{\alpha_i \omega_j} \right)^{\alpha_i - 1} \right) + n_j \frac{\omega_i}{\alpha_i - 1} \right).$$

□

6.C Postponed tables of Section 6.6

$m=2$	α_1	1.5	1.5	1.5	1.5	2	2	2	6	6	26
	α_2	1.5	2	6	26	2	6	26	6	26	26
ω_1	ω_2										
0.5	0.5	32.1	12.2			23.5			7.4		1.9
		1.7	1.7			1.1			0.2		
0.5	1	18.6	29.8	0.2		9.1	1.1		0.5		
		1.4	3.9	0.1		0.7	0.7		0.1		
0.5	5	2.3	5.8	26.2		0.7	6.9			0.4	
		0.3	0.2	7.8		0.1	0.2				
0.5	25	0.3	1.1	5.0	25.0		0.6	6.7			
					9.1			0.3			
1	1	**32.6**	12.0			22.3	0.1		7.5		2.0
		1.8	2.0			0.9			0.2		
1	5	6.7	13.4	5.3		2.0	18.1			5.7	
		0.5	0.3	4.1		0.2	4.6			1.8	
1	25	0.6	1.7	10.6	5.1	0.1	1.5	17.5			
			0.1	0.1	5.1			6.4			
5	5	32.3	12.6			23.5			7.4		1.9
		1.9	1.9			1.1			0.2		
5	25	5.9	11.0	5.8		2.5	17.9			6.0	
		0.7	0.4	4.4		0.2	4.4			1.8	
25	25	30.6	12.8			23.0			7.3		2.0
		2.2	2.1			1.1			0.2		

Table 6.2: Relative deviation of SEPT and ℓ-SEPT from OPT in percentages for 2 job classes. The upper value in a cell is the deviation for SEPT, the lower one is the deviation for ℓ-SEPT. In case the deviation from either SEPT or ℓ-SEPT from the optimal policy is less than 0.1%, the value has been surpressed in the table.

$m=3$		α_1	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	2	2	2	2	2	2	6
		α_2	1.5	1.5	1.5	1.5	2	2	2	6	6	26	2	2	2	6	6	26	6
		α_3	1.5	2	6	26	2	6	26	6	26	26	2	6	26	6	26	26	6
ω_1	ω_2	ω_3																	
0.5	0.5	0.5	48.7	30.0	28.8	33.6	23.0	9.6	12.6	1.6		0.1	33.0	17.1	21.6	2.2		0.2	10.4
			2.7	2.9	1.4	1.5	2.4	1.5	1.8				1.5	0.8	1.0				0.3
0.5	0.5	1	36.1	48.8	24.3	30.7	28.9	9.6	11.4	0.2			20.1	15.8	19.9	0.9			3.5
			2.1	4.7	1.4	1.5	3.5	1.5	1.7	0.1			1.2	1.2	0.9	0.5			0.1
0.5	0.5	5	10.7	16.9	40.5	26.2	7.6	23.3	8.8	22.4			5.6	17.1	14.2	6.3	0.2		1.6
			0.6	0.7	8.8	1.3	0.6	5.6	1.3	6.7			0.3	0.6	0.7	0.1			
0.5	0.5	25	2.6	4.2	15.6	42.2	1.9	7.2	23.3	3.6	20.9	24.1	1.4	5.2	16.3	0.6	6.7	7.0	0.5
			0.2	0.2	0.6	**10.5**	0.1	0.5	7.0		8.5	9.2		0.2	0.7		0.3	0.3	
0.5	1	1	36.2	27.3	15.0	17.5	39.7	22.9	30.4	2.4	0.1	0.2	22.2	6.8	9.2	4.0	0.8	0.4	5.0
			2.3	3.1	0.9	1.3	4.5	3.1	3.6	0.3	0.1		1.3	0.7	0.6	0.8	0.5		0.2
0.5	1	5	10.1	17.5	22.6	15.5	13.4	34.7	21.8	22.0	1.8		4.4	19.1	6.9	5.8	2.8		0.1
			0.8	0.8	6.2	1.0	1.3	7.8	3.0	5.9	0.6		0.4	3.1	0.5	0.3	1.0		
0.5	1	25	2.1	3.8	15.6	21.4	3.6	11.2	35.6	3.8	17.8	23.1	0.8	3.8	18.2	0.6	5.4	6.1	
			0.2	0.2	0.5	7.1	0.4	1.1	9.5	0.1	7.4	9.0		0.2	4.3		0.4	0.4	
0.5	5	5	30.4	13.6	7.0	2.9	22.2	10.2	4.4	26.3	21.0	0.6	20.7	2.3	0.7	11.0	4.9	0.8	6.8
			1.7	1.9	1.7	0.2	1.0	2.2	0.2	7.3	6.0		0.9			0.3	0.1		0.2
0.5	5	25	6.1	10.6	8.3	6.6	3.6	18.6	8.8	9.8	26.8	19.4	2.4	15.8	2.3	1.9	9.3	5.1	
			0.6	0.4	4.2	2.1	0.3	3.7	2.7	2.2	8.7	7.3	0.2	4.0	0.1		1.3	0.3	
0.5	25	25	29.4	12.5	0.7	1.8	22.0	1.4	2.5	9.3	9.2	22.6	22.1	0.2	0.5	7.0	1.8	6.8	7.5
			2.0	2.0		0.6	0.9		1.0	0.2	2.6	8.4	1.1			0.2		0.3	0.2
1	1	1	**51.0**	31.6	28.7	32.2	24.2	10.3	11.5	1.6		0.1	33.9	17.7	21.8	2.2		0.2	10.4
			2.4	2.8	1.6	1.6	2.6	1.6	1.8				1.5	0.8	1.0	0.1			0.3
1	1	5	17.8	25.5	24.6	29.7	13.4	16.9	10.8	4.4	1.3		9.5	28.3	17.9	14.9	1.7		2.2
			1.3	1.0	4.7	1.3	1.0	5.0	1.5	3.3	0.4		0.6	4.8	0.8	3.4	0.5		
1	1	25	5.1	8.3	25.2	23.8	3.2	12.1	16.0	9.3	4.1	4.7	2.7	9.1	28.4	1.6	13.2	16.1	0.8
			0.3	0.4	0.7	5.6	0.3	0.6	6.3	0.1	4.1	4.7	0.1	0.3	6.5		5.0	6.0	
1	5	5	29.7	16.7	6.1	5.9	25.8	10.6	11.4	8.6	5.0	0.4	19.8	6.7	2.1	20.0	14.5	0.6	6.0
			2.0	1.9	1.6	0.5	1.0	1.6	0.4	4.4	3.9		1.0	1.4	0.2	4.2	3.4		0.1
1	5	25	7.2	11.3	10.3	5.7	5.5	21.4	10.5	8.5	8.2	4.5	2.7	15.0	6.1	6.0	18.7	13.3	
			0.7	0.5	3.7	1.7	0.3	3.4	1.9	1.5	5.9	4.5	0.2	3.7	1.9	1.2	5.9	4.9	
5	5	5	48.1	31.1	27.7	30.2	23.1	10.4	11.9	1.6		0.1	33.7	18.0	21.7	2.3		0.2	10.5
			2.7	2.8	1.5	1.6	2.5	1.6	1.9				1.5	0.8	1.0	0.1			0.3

Table 6.3: Relative deviation of SEPT (upper cell) and ℓ-SEPT (lower cell) from OPT in percentages for 3 job classes. An entry has been suppressed in case it is less than 0.1%. Also, settings for which either $(\alpha_1, \alpha_2, \alpha_3)$ is lexicographically bigger than $(6, 6, 6)$ or $(\omega_1, \omega_2, \omega_3)$ is lexicographically bigger than $(5, 5, 5)$ have been suppressed.

$m=$ 5				α_1	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5
				α_2	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	26
				α_3	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	26	26
				α_4	1.5	1.5	1.5	1.5	2	2	2	6	6	26	26	26
				α_5	1.5	2	6	26	2	6	26	6	26	26	26	26
ω_1	ω_2	ω_3	ω_4	ω_5												
0.5	0.5	0.5	0.5	0.5	68.8	53.9	57.1	61.8	42.8	39.7	45.1	41.0	45.3	47.8	29.2	0.7
					3.5	3.8	2.8	2.6	3.6	2.9	3.2	2.0	2.2	2.6	1.6	0
0.5	0.5	0.5	0.5	1	58.1	68.2	51.9	59.5	52.6	38.5	44.8	36.6	43.9	48.3	30.7	0.3
					3.0	5.1	2.7	2.8	4.7	2.6	3.3	2.0	2.3	2.3	1.5	0
0.5	0.5	0.5	0.5	5	29.3	38.1	62.3	53.2	28.1	49.0	37.3	50.4	38.2	39.6	24.9	0.2
					1.6	1.7	10.2	2.4	1.7	7.5	2.7	7.8	1.9	2.0	1.2	0
0.5	0.5	0.5	0.5	25	9.8	14.8	38.4	60.7	11.7	25.5	47.6	24.8	49.2	53.1	41.6	23.1
					0.6	0.7	1.6	**12.5**	0.7	1.5	8.8	0.9	9.5	10.5	10.1	8.4
0.5	0.5	0.5	1	1	50.1	50.1	37.6	42.8	63.2	49.5	57.0	34.9	37.4	45.8	25.1	0.3
					3.0	4.4	2.3	2.4	6.1	4.3	5.1	1.8	2.2	2.2	1.2	0
0.5	0.5	0.5	1	5	23.7	31.0	46.7	39.4	34.7	56.6	51.0	45.2	33.6	40.5	22.1	0
					1.7	1.6	7.0	2.1	2.8	8.9	4.3	7.6	1.8	2.0	1.2	0
0.5	0.5	0.5	1	25	8.8	13.2	31.8	46.8	15.2	33.3	57.0	24.4	45.2	52.5	39.5	22.4
					0.6	0.7	1.4	8.4	1.3	2.5	10.3	0.9	8.9	9.8	9.8	8.3
0.5	0.5	0.5	5	5	33.5	22.5	23.3	17.8	30.3	31.7	23.5	51.8	47.0	34.5	17.6	0.6
					1.8	1.8	3.7	1.0	1.3	4.2	1.2	9.9	7.4	1.6	0.9	0
0.5	0.5	0.5	5	25	9.8	14.9	16.8	24.5	10.0	26.4	29.9	29.5	51.6	45.5	34.1	17.8
					0.9	0.7	3.3	4.2	0.6	3.0	5.2	4.2	11.4	8.9	8.5	6.7
0.5	0.5	0.5	25	25	29.4	14.6	6.0	9.4	23.4	7.8	14.8	18.8	30.4	48.1	37.6	23.0
					1.8	2.0	0.3	1.7	1.0	0.4	2.5	0.6	4.9	11.7	10.9	8.1
0.5	0.5	25	25	25	43.7	28.9	24.6	29.7	22.7	11.1	13.6	4.3	4.6	7.1	32.7	20.2
					2.9	3.0	1.5	2.2	2.5	1.7	2.4	0.1	0.9	2.0	9.9	6.9
0.5	25	25	25	25	54.2	40.5	39.1	43.3	31.3	25.0	28.1	21.6	23.9	27.4	4.3	18.0
					3.5	3.6	2.5	2.9	3.4	2.5	3.0	1.3	1.8	2.0	1.4	5.9
1	1	1	1	1	**69.8**	54.4	58.3	60.0	44.2	42.0	45.1	43.4	43.5	50.6	29.4	0.6
					3.6	3.8	3.2	2.9	3.7	3.0	3.4	2.0	2.2	2.4	1.5	0

Table 6.4: Relative deviation of SEPT (upper cell) and ℓ-SEPT (lower cell) from OPT in percentages for 5 job classes. Many settings for which either $(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5)$ is lexicographically bigger than $(1.5, 1.5, 1.5, 26, 26)$ or $(\omega_1, \omega_2, \omega_3, \omega_4, \omega_5)$ is lexicographically bigger than $(0.5, 0.5, 0.5, 25, 25)$ have been suppressed to reduce the table size. In general, for these setting both SEPT and ℓ-SEPT perform well, that is, better than many of the values given in the table.

Bibliography

[1] E.H.L. Aarts and J.K. Lenstra. *Local Search in Combinatorial Optimization*. Wiley, Chichester, United Kingdom, 1997.

[2] K. Albers and F. Slomka. An event stream driven approximation for the analysis of real-time systems. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 187–195, 2004.

[3] S. Anand, N. Garg, and N. Megow. Meeting deadlines: How much speed suffices? In G. Ausiello and V. Sassone, editors, *Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 232–243, 2011.

[4] V.F. Araman and R. Caldentey. Dynamic pricing for nonperishable products with demand learning. *Operations Research*, 57(5):1169–1188, 2009.

[5] J. Aspnes, Y. Azar, A. Fiat, S.A. Plotkin, and O. Waarts. Online routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the Association for Computing Machinery (ACM)*, 44(3):486–504, 1997.

[6] B. Awerbuch, Y. Azar, E. Grove, M. Kao, P. Krishnan, and J. Vitter. Load balancing in the l_p norm. In *Proceedings of the 36th Symposium on Foundations of Computer Science (FOCS)*, pages 383–391, 1995.

[7] B. Awerbuch, Y. Azar, Y. Richter, and D. Tsur. Tradeoffs in worst-case equilibria. *Theoretical Computer Science*, 361(2-3):200–209, 2006.

[8] Y. Azar and A. Epstein. Convex programming for scheduling unrelated machines. In *Proceedings of 37th Symposium on Theory of Computing (STOC)*, pages 331–337. Association for Computing Machinery (ACM), 2005.

[9] K.S. Azoury. Bayes solution to dynamic inventory models under unknown demand distribution. *Management Science*, 31(9):1150–1160, 1985.

[10] T.P. Baker and S.K. Baruah. Schedulability analysis of multiprocessor sporadic task systems. In I. Lee, J.Y.-T. Leung, and S.H. Sou, editors, *Handbook of Real-Time and Embedded Systems*, chapter 3. CRC Press, 2007.

[11] S. Baruah and N. Fisher. The partitioned multiprocessor scheduling of sporadic task systems. In *Proceedings of 26th IEEE Real-Time Systems Symposium*, pages 321–329. IEEE, 2005.

[12] S.K. Baruah. The partitioned EDF scheduling of sporadic task systems. In *Proceedings of 32nd IEEE Real-Time Systems Symposium*, pages 116–125. IEEE, 2011.

[13] S.K. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, and S. Stiller. Improved multiprocessor global schedulability analysis. *Real-Time Systems*, 46(1):3–24, 2010.

[14] S.K. Baruah, N. Cohen, C.G. Plaxton, and D. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, 1996.

[15] S.K. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th IEEE Real-Time Systems Symposium*, pages 182–190. IEEE, 1990.

[16] L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, G. Schäfer, and T. Vredeveld. Average case and smoothed competitive analysis for the multi-level feedback algorithm. *Mathematics of Operations Research*, 31(3):85–108, 2006.

[17] R. Beier, H. Röglin, and B. Vöcking. The smoothed number of pareto optimal solutions in bicriteria integer optimization. In *Proceedings of the 12th international conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 53–67, Berlin, Heidelberg, 2007. Springer-Verlag.

[18] R. Beier and B. Vöcking. Random knapsack in expected polynomial time. *Journal of Computer and System Sciences*, 69(3):306–329, 2004.

[19] M. Bertogna, M. Cirinei, and G. Lipari. Improved schedulability analysis of EDF on multiprocessor platforms. In *Proceedings of 17th Euromicro Conference on Real-Time Systems*, pages 209–218. IEEE, 2005.

[20] F. Bock. An algorithm for solving traveling-salesman and related network optimization problems. In *Proceedings of the 14th National Meeting of the Operations Research Society of America (ORSA)*, St. Louis, Missouri, Unites States of America, 1958.

[21] V. Bonifaci, A. Marchetti-Spaccamela, and S. Stiller. A constant-approximate feasibility test for multiprocessor real-time scheduling. *Algorithmica*, 62(3-4):1034–1049, 2012.

[22] V. Bonifaci and A. Wiese. Scheduling unrelated machines of few different types. Unpublished manuscript, available at http://arxiv.org/abs/1205.0974, 2012.

[23] P. Brucker, J. Hurink, and F. Werner. Improving local search heuristics for some scheduling problems - II. *Discrete Applied Mathematics*, 72(1-2):47–69, 1997.

[24] T. Brunsch, H. Röglin, C. Rutten, and T. Vredeveld. Smoothed performance guarantees for local search. In *European Symposium on Algorithms (ESA)*, volume 6942 of *Lecture Notes in Computer Science*, pages 772–783. Springer, Berlin, 2011.

[25] L. Buriol, M. Ritt, F. Rodrigues, and G. Schäfer. On the smoothed price of anarchy of the traffic assignment problem. In *11th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, volume 20, pages 122–133, Dagstuhl, Germany, 2011. Schloss Dagstuhl, Leibniz-Zentrum für Informatik.

[26] A.N. Burnetas and M.N. Katehakis. On sequencing two types of tasks on a single processor under incomplete information. *Probability in the Engineering and Informational Sciences*, 7(1):85–119, 1993.

[27] G. Centeno and R.L. Armacost. Minimizing makespan on parallel machines with release time and machine eligibility restrictions. *International Journal of Production Research*, 42(6):1243–1256, 2004.

[28] S. Chakraborty, S. Künzli, and L. Thiele. Approximate schedulability analysis. In *Proceedings of 23rd IEEE Real-Time Systems Symposium*, pages 159–168. IEEE, 2002.

[29] C. Chekuri and S. Khanna. On multi-dimensional packing problems. *SIAM Journal on Computing*, 33(4):837–851, 2004.

[30] J.-J. Chen and S. Chakraborty. Resource augmentation bounds for approximate demand bound functions. In *Proceedings of 32nd IEEE Real-Time Systems Symposium*, pages 272–281. IEEE, 2011.

[31] J.-J. Chen and S. Chakraborty. Partitioned packing and scheduling for sporadic real-time tasks in identical multiprocessor systems. Presented at the 24th Euromicro Conference on Real-Time Systems, Paper received through personal communication, 2012.

[32] L. Chen and E.L. Plambeck. Dynamic inventory management with learning about the demand distribution and substitution probability. *Manufacturing & Service Operations Management*, 10(2):236–256, 2008.

[33] Y. Cho and S. Sahni. Bounds for list schedules on uniform processors. *SIAM Journal on Computing*, 9(1):91–103, 1980.

[34] C. Chung, T. Nonner, and A. Souza. SRPT is 1.86-competitive for completion time scheduling. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1373–1388, 2010.

[35] S.A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual Symposium on the Theory of Computing (STOC)*, pages 151–158. Association for Computing Machinery (ACM), 1971.

[36] E. Cope. Bayesian strategies for dynamic pricing in e-commerce. *Naval Research Logistics*, 54(3):265–281, 2006.

[37] G.A. Croes. A method for solving traveling salesman problems. *Operations Research*, 6(6):791812, 1958.

[38] A. Czumaj and B. Vöcking. Tight bounds for worst-case equilibria. *Association for Computing Machinery (ACM) Transactions on Algorithms*, 3(1):4th article, 2007.

[39] B.C. Dean. *Approximation Algorithms for Stochastic Scheduling Problems*. PhD thesis, Massachusetts Institute of Technology, 2005.

[40] M.H. DeGroot. *Optimal Statistical Decisions*. McGraw-Hill, New York, NY, United States of America, 1970.

[41] T. Ebenlendr, M. Krcal, and J. Sgall. Graph balancing: A special case of scheduling unrelated parallel machines. In *Proceedings 19th Symposium on Discrete Algorithms (SODA)*, pages 483–490, 2008.

[42] F. Eisenbrand and T. Rothvoss. A PTAS for static priority real-time scheduling with resource augmentation. In *Proceedings of 35th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 246–257, 2008.

[43] F. Eisenbrand and T. Rothvoss. EDF-schedulability of synchronous periodic task systems is co-np-hard. In *Proceedings of 21st Symposium On Discrete Algorithms (SODA)*, pages 1029–1034, 2010.

[44] M. Englert, H. Röglin, and B. Vöcking. Worst case and probabilistic analysis of the 2-opt algorithm for the TSP. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1295–1304, 2007.

[45] F.F. Farias and B. Van Roy. Dynamic pricing with a prior on market response. *Operations Research*, 58(1):16–29, 2010.

[46] R. Feldmann, M. Gairing, T. Lücking, B. Monien, and M. Rode. Nashification and the coordination ratio for a selfish routing game. In *Proceedings of 30th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 2719 of *Lecture Notes in Computer Science*, pages 514–526. Springer, Heidelberg (2003), 2003.

[47] G. Finn and E. Horowitz. A linear time approximation algorithm for multiprocessor scheduling. *BIT Numerical Mathematics*, 19(3):312–320, 1979.

[48] N. Fisher, S. Baruah, and T.P. Baker. The partitioned scheduling of sporadic tasks according to static-priorities. In *Proceedings of the 18th Euromicro Conference on Real-Time Systems*, pages 118–127, 2006.

[49] M. Gairing, T. Lücking, M. Mavronicolas, and B. Monien. The price of anarchy for restricted parallel links. *Parallel Processing Letters*, 16(1):117–131, 2006.

[50] M. Gairing, T. Lücking, M. Mavronicolas, and B. Monien. Computing nash equilibria for scheduling on restricted parallel links. *Theory of Computing Systems*, 47(2):405–432, 2010.

[51] M. Gairing, T. Lücking, M. Mavronicolas, B. Monien, and P. Spirakis. Structure and complexity of extreme nash equilibria. *Theoretical Computer Science*, 343(1-2):133–157, 2005.

[52] H.L. Gantt. Work, wages and profit. *The Engineering Magazine, New York*, 1910. Republished as Work, Wages and Profits, Easton, Pennsylvania, Hive Publishing Company, 1974, ISBN 0879600489.

[53] M.R. Garey and D.S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, 4(4):397–411, 1975.

[54] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman & Co., New York, NY, United States of America, 1979.

[55] J.C. Gittins. Bandit processes and dynamic allocation indices (with discussion). *Journal of the Royal Statistical Society, Series B (Methodological)*, 41(2):148–177, 1979.

[56] J.C. Gittins. *Multi-armed bandit allocation indices*. Wiley, New York, NY, United States of America, 1989.

[57] J.C. Gittins and K.D. Glazebrook. On Bayesian models in stochastic scheduling. *Journal of Applied Probability*, 14(3):556–565, 1977.

[58] J.C. Gittins and D.M. Jones. A dynamic allocation index for the sequential design of experiments. In *Progress in Statistics*, pages 241–266. North-Holland, The Netherlands, 1974.

[59] K.D. Glazebrook and R.W. Owen. On the value of adaptive solutions to stochastic scheduling problems. *Mathematics of Operations Research*, 20(1):65–89, 1995.

[60] F. Glover and G. Kochenberger. *Handbook of Metaheuristics*. Kluwer Academic Publishers, Norwell, Massachusetts,United States of America, 2003.

[61] J. Goossens, S. Funk, and S.K. Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Systems*, 25(2-3):187–205, 2003.

[62] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[63] T. Hamada and K.D. Glazebrook. A Bayesian sequential single machine scheduling problem to minimize the expected weighted sum of flowtimes of jobs with exponential processing times. *Operations Research*, 41(5):924–934, 1993.

[64] T. Hamada and M. Tamaki. Some results on a Bayesian sequential scheduling on two identical parallel processors. *Journal of the Operations Research Society of Japan*, 42(14):316–329, 1999.

[65] D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the Association for Computing Machinery (ACM)*, 34(1):144–162, 1987.

[66] M. Hoefer and A. Souza. Tradeoffs and average-case equilibria in selfish routing. *ACM Transactions on Computation Theory*, 2(1):2nd article, 2010.

[67] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[68] H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann / Elsevier, Amsterdam, The Netherlands, 2004.

[69] C.A.J. Hurkens and T. Vredeveld. Local search for multiprocessor scheduling: how many moves does it take to a local optimum? *Operations Research Letters*, 31(2):137–141, 2003.

[70] N. Immorlica, L. Li, V. Mirrokni, and A.S. Schulz. Coordination mechanism for selfish scheduling. *Theoretical Computer Science*, 410(17):1589–1598, 2009.

[71] K. Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

[72] K. Jansen and L. Porkolab. Improved approximation schemes for scheduling unrelated parallel machines. In *Proceedings of 31st Symposium on Theory of Computing (STOC)*, pages 408–417, 1999.

[73] N. Karmarkar and R.M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 312–320, 1982.

[74] R.M. Karp, F.T. Leighton, R.L. Rivest, C.D. Thompson, U.V. Vazirani, and V.V. Vazirani. Global wire routing in two-dimensional arrays. *Algorithmica*, 2(1):113–129, 1987.

[75] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. In *Proceedings of the 16th annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1563 of *Lecture Notes in Computer Science*, pages 404–413, 1999. Springer, Berlin.

[76] V.S. Anil Kumar, M.V. Marathe, S. Parthasarathy, and A. Srinivasan. Approximation algorithms for scheduling on multiple machines. In *Proceedings of 46th Symposium on Foundations of Computer Science (FOCS)*, pages 254–263. IEEE, 2005.

[77] M.A. Lariviere and E.L. Porteus. Stalking information: Bayesian inventory management with unobserved lost sales. *Management Science*, 45(3):346–363, 1999.

[78] J.K. Lenstra, D.B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46(3):259–271, 1990.

[79] J.Y.T. Leung and C.L. Li. Scheduling with processing set restrictions: A survey. *International Journal of Production Economics*, 116(2):251–262, 2008.

[80] C.L. Li. Scheduling unit-length jobs with machine eligibility restrictions. *European Journal of Operational Research*, 174(2):1325–1328, 2006.

[81] L.W. Liao and G.J. Sheen. Parallel machine scheduling with machine availability and eligibility constraints. *European Journal of Operational Research*, 184(2):458–467, 2008.

[82] K.Y. Lin. Dynamic pricing with real-time demand learning. *Operations Research*, 174(1):522–538, 2003.

[83] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the Association for Computing Machinery (ACM)*, 20(1):46–61, 1973.

[84] P. Lu and C. Yu. Worst-case nash equilibria in restricted routing. In *Proceedings of the 4th International Workshop on Internet and Network Economics (WINE)*, volume 5385 of *Lecture Notes in Computer Science*, pages 231–238, 2008.

[85] X. Lu, J.-S. Song, and K. Zhu. Inventory control with unobservable lost sales and Bayesian updates. Working paper, 2006.

[86] S. Marbán, C. Rutten, and T. Vredeveld. Learning in stochastic machine scheduling. In C. Kaklamanis and M. Skutella, editors, *Proceedings of the 9th Workshop of Approximation and Online Algorithms (WAOA) 2011*, volume 7164 of *Lecture Notes in Computer Science*, pages 21–34. Springer, Berlin, 2012.

[87] A. Marchetti-Spaccamela, C. Rutten, S. van der Ster, and A. Wiese. Assigning sporadic tasks to unrelated parallel machines. In *Proceedings of the 39th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 7391 of *Lecture Notes in Computer Science*, pages 665–676. Springer, Berlin, 2012.

[88] N. Megow, M. Uetz, and T. Vredeveld. Models and algorithms for stochastic online scheduling. *Mathematics of Operations Research*, 31(3):513–525, 2006.

[89] N. Megow and T. Vredeveld. Approximation results for preemptive stochastic online scheduling. In *European Symposium on Algorithms (ESA) 2006*, pages 516–527, 2006. Lecture Notes in Computer Science; Vol. 4168.

[90] W. Michiels, E. Aarts, and J. Korst. *Theoretical Aspects of Local Search*. Springer-Verlag, Berlin, Germany, 2007.

[91] R.H. Möhring, F.J. Radermacher, and G. Weiss. Stochastic scheduling problems i: General strategies. *Mathematical Methods of Operations Research*, 28(7):193–260, 1984.

[92] R.H. Möhring, F.J. Radermacher, and G. Weiss. Stochastic scheduling problems ii: Set strategies. *Mathematical Methods of Operations Research*, 29(3):A65–A104, 1985.

[93] R.H. Möhring, A.S. Schulz, and M. Uetz. Approximation in stochastic scheduling: the power of LP-based priority policies. *Journal of the Association for Computing Machinery (ACM)*, 46(6):924–942, 1999.

[94] J. Ou, J.Y.-T. Leung, and C.L. Li. Scheduling parallel machines with inclusive set restrictions. *Naval Research Logistics*, 55(4):328–338, 2008.

[95] C.A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32(2):163–200, 2002.

[96] C.A. Phillips, C. Stein, and J. Wein. Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82(1-2):199 – 223, 1998.

[97] A.C. Pigou. *The Economics of Welfare*. Macmillan and Co, London, United Kingdom, 1920.

[98] D. Recalde, C. Rutten, P. Schuurman, and T. Vredeveld. Local search performance guarantees for restricted related parallel machine scheduling. In *Proceedings of LATIN 2010: Theoretical Informatics*, volume 6034 of *Lecture Notes in Computer Science*, pages 108–119. Springer, Berlin, 2010.

[99] U. Rieder and J. Weishaupt. Customer scheduling with incomplete information. *Probability in the Engineering and Informational Sciences*, 9(2):269–284, 1995.

[100] M.H. Rothkopf. Scheduling with random service times. *Management Science*, 12(9):703–713, 1966.

[101] T. Roughgarden. The price of anarchy is independent of the network topology. *Journal of Computer and System Sciences*, 67(2):341–364, 2003.

[102] T. Roughgarden and É. Tardos. How bad is selfish routing? *Journal of the Association for Computing Machinery (ACM)*, 49(2):236 – 259, 2002.

[103] C. Rutten, D. Recalde, P. Schuurman, and T. Vredeveld. Performance guarantees of jump neighborhoods on restricted related parallel machines. *Operations Research Letters*, 40(4):287–291, 2012.

[104] H. Scarf. Bayes solutions of the statistical inventory problem. *The Annals of Mathematical Statistics*, 30(2):490–508, 1959.

[105] G. Schäfer. Lecture notes of the course algorithmic game theory. Online available at http://homepages.cwi.nl/∼schaefer/courses/vu-agt10/script-vu-agt10.pdf, VU University Amsterdam, 2010.

[106] G. Schäfer and N. Sivadasan. Topology matters: Smoothed competitiveness of metrical task systems. *Theoretical Computer Science*, 341(1-3):3–14, 2005.

[107] L. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16(3):687 – 690, 1968.

[108] A.S. Schulz. New old algorithms for stochastic scheduling. In *Algorithms for Optimization with Incomplete Information, Dagstuhl Seminar Proceedings, Vol. 05031*, Dagstuhl, Germany, 2005. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[109] P. Schuurman and T. Vredeveld. Performance guarantees of local search for multiprocessor scheduling. *INFORMS Journal of Computing*, 19(1):52–63, 2007.

[110] R.A. Sitters. Competitive analysis of preemptive single-machine scheduling. *Operations Research Letters*, 38(6):585–588, 2010.

[111] W.E. Smith. Various optimizers for single stage production. *Naval Research Logistics Quaterly*, 3(1-2):59–66, 1956.

[112] D.A. Spielman and S.H. Teng. Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. *Journal of the Association for Computing Machinery (ACM)*, 51(3):385–463, 2004.

[113] D.A. Spielman and S.H. Teng. Smoothed analysis: an attempt to explain the behavior of algorithms in practice. *Communications of the Association for Computing Machinery (ACM)*, 52(10):76–84, 2009.

[114] O. Svensson. Santa claus schedules jobs on unrelated machines. In *Proceedings of the 43rd Symposium on Theory of Computing (STOC)*, pages 617–626. Association for Computing Machinery (ACM), 2011.

[115] G.L. Vairaktarakis and X. Cai. The value of processing flexibility in multipurpose machines. *IIE Transactions*, 35(8):763–774, 2003.

[116] B. Vöcking. Selfish load balancing. In N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory*, chapter 20. Cambridge University Press, New York, NY, United States of America, 2007.

[117] G. Weiss. Approximation results in parallel machines stochastic scheduling. *Annals of Operations Research*, 26(1):195–242, 1990.

[118] G. Weiss. Turnpike optimality of Smith's rule in parallel machines stochastic scheduling. *Mathematics of Operations Research*, 17(2):255–270, 1992.

Nederlandse Samenvatting

Dit proefschrift bestudeert planningsproblemen, in de literatuur beter bekend als machine-volgorde problemen. Bij deze problemen moet men denken aan een groep machines die gezamenlijk een verzameling aan taken moet uitvoeren. De productietijd van een taak is de tijd die een machine nodig heeft om de taak af te werken en kan verschillen per taak en machine combinatie. Het beslissingsprobleem is welke machine welke taken moet verwerken zodat bijvoorbeeld de laatste machine zo vroeg mogelijk klaar is met haar laatste taak. Een ander voorbeeld is waarbij men de som van completeringstijden over alle taken wil minimaliseren.

Deze machine-volgorde problemen komen veelvuldig voor in de praktijk: van de manager van een autogarage die bepaalt welke werknemer welke auto repareert, tot de productieplanner van grote industriële productielijnen die de producten bepaalt die iedere productielijn de komende week gaat maken. In de literatuur is er veel onderzoek gedaan naar het ontwerpen van goede strategieën, ofwel algoritmes, voor een groot scala aan machine-volgorde problemen. Vanwege de complexiteit van deze problemen is het vaak niet mogelijk een algoritme te ontwerpen dat altijd het best mogelijke productieschema vindt binnen een acceptabele rekentijd. Daarom trachten wetenschappers meestal algoritmes te ontwikkelen die wel binnen acceptabele rekentijd een goed productieschema, ook wel oplossing genoemd, teruggeven. Een dergelijk algoritme wordt een approximatief algoritme genoemd als het, voor ieder mogelijke inputdata, een oplossing teruggeeft die bewijsbaar dichtbij de best mogelijke oplossing voor die inputdata ligt. In detail betekent dit dat de waarde van de geretourneerde oplossing maximaal een bepaalde absolute of relatieve afwijking heeft ten op zichte van de waarde van de optimale oplossing.

Vele ontwikkelde approximatieve strategieën zijn echter niet altijd geschikt voor implementatie in de praktijk. Dat deze weldoordachte algoritmes vaak de werkvloer niet bereiken kan zijn vanwege velerlei redenen; het duurt te lang voordat ze een oplossing retourneren, ze zijn te moeilijk in gebruik en begrip, de noodzakelijke data is niet voorhanden, ze werken alleen onder labcondities terwijl in de praktijk allerlei randfactoren meespelen, etc. Het doel van dit proefschrift is dan ook het onderzoeken van de kracht van simpele en snelle algoritmes die wel makkelijk in de praktijk geïmplementeerd kunnen worden. In het bijzonder bestudeer ik de kwaliteit van de schema's die geretourneerd worden door een aantal simpele algoritmes voor verschillende machine-volgorde problemen en daarbij bekijk ik hoe die kwaliteit zich verhoudt ten opzichte van de kwaliteit van het optimale schema.

In Hoofdstuk 2 onderzoek ik lokale zoekmethoden waarbij het algoritme zich van oploss-

ing naar oplossing beweegt en pas stopt als vanuit een huidige oplossing geen nieuwe betere buuroplossing kan worden gecreëerd. Een buuroplossing van een huidige oplossing wordt gecreëerd door een mutatie van de huidige oplossing: een taak wordt verwezen naar een andere machine. Ik bestudeer twee verschillende regels om een taak toe te wijzen aan een andere machine, de zogenoemde jump- en lexjump-buurruimte. Het specifieke probleem dat ik bestudeer kent m machines waaraan n taken moeten worden toegekend. De taken hebben verschillende groottes en de machines hebben verschillende snelheden zodat de productietijd van een taak gelijk is aan de taakgrootte gedeeld door de snelheid van de machine. Daarbij zijn sommige taken beperkt in de zin dat ze niet aan alle machines kunnen worden toegewezen. Ik laat zien dat de kwaliteit van een geretourneerde oplossing van de jump- en lexjump-buurruimte respectievelijk maximaal een factor $\sqrt{ms}$ en $c * \log S / \log\log S$ afliggen van de kwaliteit van de optimale oplossing. Hierbij, stelt s de maximale snelheid over alle machines voor, S de som van alle snelheden en c is een voldoende grote constante.

Hoofdstuk 3 bekijkt hetzelfde machine-volgorde probleem en ook dezelfde algoritmes als het voorgaande hoofdstuk. Dit hoofdstuk bestudeert de kwaliteit van de oplossingen van beide algoritmes als de invoerdata voor het algoritme onderhevig is aan een kleine hoeveelheid ruis. De motivatie is afgeleid van het feit dat algoritmes in praktijk het vaak veel beter doen dan de theoretische slechtste-geval-garantie. Deze populaire theoretische garantie bekijkt de kwaliteit van een algoritme voor de slechts mogelijke invoerdata en zegt daarmee dat voor een andere invoer de kwaliteit nooit slechter kan zijn dan de waarde van deze slechtste-geval-garantie. Die slechts mogelijke invoerdata is echter vaak kunstmatig geconstrueerd en wordt zelden of nooit waargenomen in de praktijk. Het gat tussen de kwaliteit van een algoritme geobserveerd in praktijk en de theoretische slechtste-geval-garantie is dan ook vaak groot. Hoofdstuk 3 tracht dit gat te verkleinen door een kleine hoeveelheid ruis toe te voegen aan de invoerdata in de hoop de onrealistische structuur van de slechts mogelijke invoerdata te verstoren en zodoende weer een goede kwaliteit oplossing te bereiken. Het hoofdstuk laat inderdaad zien dat, in de aanwezigheid van een beetje ruis, de kwaliteit van de bestudeerde algoritmes significant verbeterd ten opzichte van de slechtste-geval-garanties die werden afgeleid in Hoofdstuk 2.

Hoofdstukken 4 en 5 bekijken een alternatief machine-volgorde probleem waarbij de centrale vraag is of er überhaupt een productieschema bestaat dat aan alle beperkingen en taakeigenschappen voldoet. Iedere taak brengt over tijd met een vast interval een serie aan kleinere subtaken voort met dezelfde productietijd maar met ieder een eigen relatieve deadline. Na een verdeling van de taken over machines wordt vervolgens op iedere machine het zogenoemde Vroegste-Deadline-Eerst (VDE) algoritme toegepast om tot een productieschema per machine te komen. VDE is een simpel en populair algoritme dat optimaal is voor het geval dat er slechts een machine beschikbaar is. Het resulterende vraagstuk voor meerdere machines is het volgende: bestaat er een toewijzing van de taken aan de machines zodat, als VDE wordt toegepast per machine, een valide productieschema per machine wordt gevonden. Hierbij definieer ik een valide productieschema als een schema waarbij iedere subtaak kan worden voltooid voor haar deadline en waarbij ook iedere subtaak wordt toegewezen aan de machine waarop de 'moedertaak' is ingedeeld.

In Hoofdstuk 4 richt ik mij op de situatie waarin de productietijd van een taak afhankelijk is van zowel de taak als de machine waarop zij wordt ingedeeld. Ik stel een 12.9-approximatief algoritme voor, dat is, het algoritme retourneert of (i) er een verdeling bestaat van de taken

over machines zodat VDE een valide productieschema per machine geeft als de machines 12.9 keer zo snel gaan, of (ii) concludeert dat er geen verdeling van de taken over de machines bestaat waarvoor VDE een valide productieschema geeft per machine als de machines hun originele snelheid behouden. De looptijd van dit algoritme is polynomiaal in de hoeveelheid invoerdate. Het tweede gedeelte van dit hoofdstuk stelt een $(1 + \epsilon)$-approximatief algoritme voor waarvan de looptijd polynomiaal is in het aantal taken maar wel exponentieel is in het aantal machines en in de constante $1/\epsilon$.

Het probleem dat bekeken wordt in Hoofdstuk 5 is een iets eenvoudigere versie van het probleem dat in Hoofdstuk 4 centraal staat. Nu is de productietijd van een taak gelijk over alle machines. Ik presenteer opnieuw een $(1+\epsilon)$-approximatief algoritme waarvan de looptijd deze keer polynomiaal is in het aantal taken en in het aantal machines en exponentieel is in slechts de constante $1/\epsilon$ en in $\log \lambda$, waar λ de ratio is van grootste relatieve deadline over de kleinste relatieve deadline.

Hoofdstuk 6 tenslotte richt zich op stochastische machine-volgorde problemen waarin de productietijd van een taak niet langer deterministisch is maar stochastisch. In standaard stochastische machine-volgorde problemen wordt aangenomen dat de onderliggende kansverdeling van een stochastische variabele volledig bekend is. In Hoofdstuk 6 verruim ik deze aanname door te stellen dat de parameters van de onderliggende distributie onbekend zijn. Ondanks dat de parameters onbekend zijn, kan men door het uitvoeren van taken en het dus observeren van realisaties van de onderliggende distributie wel leren over deze parameters. Een optimaal algoritme voor het bestuurde probleem is uit de literatuur bekend. Dit algoritme is helaas in zowel de hoeveelheid berekeningen als de noodzakelijke precisie daarvan te veeleisend om nuttig te zijn voor praktische doeleinden. In Hoofdstuk 6 bestudeer ik de kwaliteit van de oplossingen gegenereerd door twee verschillende simpele en snelle algoritmes. Ik laat zien dat het dynamische algoritme dat kan leren en zijn strategie kan aanpassen veel krachtiger is dan zijn statische variant.

Over alle hoofdstukken heen concludeer ik dat simpele algoritmes die in praktijk veel eerder toegepast zullen worden dan de reken-intensieve algoritmes uit de literatuur ook theoretisch aardig presteren. Ik heb voor een aantal van dergelijke eenvoudige algoritmes de theoretische slechtste-geval-garantie bepaald die vaak een polynomiale of soms constante waarde kent. Om de kwalitatieve analyse van algoritmes door middel van slechtste-geval-garanties dichter bij de geobserveerde kwaliteit van die algoritmes in praktijk te brengen wend ik mij tot het toepassen van het concept 'smoothing'. In smoothing wordt er een kleine hoeveelheid ruis toegevoegd aan de invoerdata om op die manier slechte maar kunstmatige instanties uit te schakelen. In de aanwezigheid van een kleine hoeveelheid ruis presteren een aantal van de besproken algoritmes nog beter en benaderen soms zelfs optimaliteit.

Curriculum Vitae

Cyriel Rutten was born on October 13, 1985 in Heerlen, The Netherlands. In 2004, he received his Atheneum diploma from the Sophianum college in Gulpen. In September of that same year he started studying Econometrics and Operations Research at Maastricht University. He specialized in Operations Research and received his Master's degree in May 2009. The master thesis was written during an internship at DSM ACES in Sittard. The next year he completed a research master in Economic and Financial research. All three mentioned degrees were obtained cum laude. In the third year of his bachelor education, Cyriel spend three months at Santa Barbara within the University of California and completed a number of courses with distinction. During his studies he was a research and teaching assistant at several occasions at Maastricht University. Further, he spend two summer internships at APG in Heerlen and also worked briefly for Mateum in Maastricht. In September 2009, he started as a PhD student in the Quantitative Economics department of Maastricht University, becoming part of the Operations Research group there. In 2011 he spend three months at La Sapienza University in Rome, Italy. The results of his research are presented in this dissertation which will be defended on the first of February 2013.